SERIE
AUTOAPRENDIZAJE
ACELERADO

colección
m/b

la mejor programación del Dragon por la práctica

DESDE LAS RUTINAS MÁS SENCILLAS A LOS PROGRAMAS MÁS ESPECTACULARES, EN UNA FORMACIÓN PROGRESIVA

Keith Brain/
Steven Brain





Título original: DRAGON 32 - GAMES MASTER

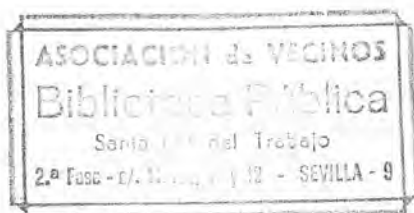
Traducción: Ramón Sangüesa Solé

(En los listados la letra Ñ ha sido substituida por /).

ISBN: 84-247-0202-6

Impreso en España — Printed in Spain

Dep. Legal: B. 490-85 — —REDEPRINT— — Barcelona



Sumario

para una tabla de ganadores. Fichero para conservar las tablas de ganadores. El efecto de perspectiva en los gráficos. La realización de mapas en la pantalla. La presentación de datos mediante gráficos. Trucos para mejorar rutinas gráficas. Creación de una ventana en el centro de la pantalla. Diseño del volante y del salpicadero de un camión. Inclusión de textos en la pantalla de alta resolución. Mezcla de textos y gráficos en la pantalla.

INTRODUCCION

El propósito principal de este libro es enseñar al lector a desarrollar sus propios juegos para el ordenador Dragón 32. No se encontrará aquí la clásica lista con refritos de juegos archiconocidos; al contrario, se sigue un método estructurado para que el lector pueda diseñar y escribir sus propios juegos. Aunque hay muchos listados completos a lo largo del libro, aconsejamos encarecidamente que se resista la tentación de conformarse tan sólo con copiarlos: estudiese a fondo el capítulo en el que se desarrolla cada programa de juego.

El libro empieza por los principios básicos y gradualmente va ganando nivel hasta llegar a conceptos de programación muy complejos. A todo el mundo le gusta jugar pero a la mayoría le disgusta seguir el proceso de aprendizaje. Sin embargo, esperamos que una vez haya asimilado el contenido del libro se sorprenda el lector al comprobar que ha aprendido acerca de la programación del Dragón 32 más de lo que esperaba.

El esquema básico que se sigue es partir de una primera idea para, paso a paso, ir desarrollándose hasta conseguir un programa interesante. En especial, nos hemos preocupado de mostrar cómo modificaciones pequeñas, producen importantes diferencias en el resultado final. En vez de indicar qué es lo que se debe o no se debe hacer, ponemos especial énfasis en enseñar el diferente resultado que se obtiene al adoptar distintos métodos de solución. En la medida de lo posible hemos querido evitar que se tengan que escribir de nuevo líneas completas. A pesar de ello, las alteraciones son frecuentes. La mayoría de las líneas tienen múltiples funciones ya que se ha considerado que las líneas de una sola función son poco eficientes y sólo para el nivel de párvulos.

Se han aprovechado todas las oportunidades para explotar al máximo las posibilidades propias del Dragón. La mayor parte de las

rutinas están acompañadas por los correspondientes diagramas de flujo para que resulte fácil comprender su funcionamiento. Todos los listados que aparecen en el libro tienen una anchura de 32 columnas. Tienen, pues, el mismo formato que la presentación en la pantalla del Dragón con la salvedad de que los caracteres en representación inversa aparecen como caracteres en minúscula. En la mayor parte de los casos se han utilizado los espacios y paréntesis generosamente con el fin de hacer los listados más legibles. Téngase en cuenta, sin embargo, que algunos de estos símbolos son imprescindibles y, por tanto, no deben borrarse todos.

Al final de cada capítulo hay una sección —«Ahora es su turno»— que permitirá al lector comprobar el grado de asimilación que ha adquirido del material explicado. En el caso de que no se consiga realizar las tareas que en esta sección se indican, el lector no debe preocuparse demasiado aunque, de todas formas, sería una buena idea que releyera el capítulo.

Esta sección no se ha diseñado con la intención de que sea un «deber» sino una forma de asegurar que realmente está aprendiéndose. Resulta también una valiosa fuente de ideas que pueden desarrollarse posteriormente.

Si usted se encuentra con problemas en sus listados o en sus ideas, hay unas cuantas técnicas de depuración de programas que resultan muy útiles. Recuerde que el Dragón tiene la función TRON que permite seguir progresivamente la ejecución de los programas. Recuérdese también que si, al mismo tiempo, se presionan las teclas SHIFT y @ , el programa se detendrá hasta que se vuelva a presionar otra tecla.

Si aun así, el problema no queda resuelto, recuérdese que las variables no se inicializan hasta que no se haga RUN o EDIT. Por tanto, se pueden imprimir sus valores y comprobar si son correctos o no. Los errores sintácticos (?SN ERROR) son normalmente resultado de la falta o el exceso de paréntesis, comillas o signos de cadenas ($) o también de entrar palabras con alguna letra errónea. Recuérdese también que pueden producirse errores cuando se eliminan espacios imprescindibles. Los errores de no concordancia de tipos ocurren cuando se confunden variables simples con cadenas (?TM ERROR).

Si se utiliza una gran cantidad de cadenas se puede recibir un mensaje de error (?OS ERROR) lo cual se arregla consiguiendo más

espacio con CLEAR. La longitud máxima de una cadena es de 255 caracteres. Si se unen dos cadenas largas se puede recibir un mensaje de error ?LS ERROR. Si se intenta inspeccionar un elemento de una matriz y este elemento no existe, se recibirá un mensaje indicando que el subíndice utilizado no es correcto (?BS ERROR). Esto ocurrirá también cuando no se haya asignado previamente ninguna longitud a la matriz —es decir se haya olvidado la instrucción DIM—; cuando la matriz sea demasiado pequeña o cuando busque elementos negativos.

Si se olvida de colocar las instrucciones NEXT o RETURN correspondientes a las FOR y GOSUB, recibirá el mensaje de error ?NF o ?RG ERROR. Finalmente, si recibe un mensaje de línea indefinida (?UL ERROR) será porque ha olvidado indicar a qué línea debía saltar el programa.

El Dragón es una fiera muy amable a la que le encantará jugar con usted. Esperamos que encuentre tanta satisfacción en el empleo de este libro como nosotros en su confección.

Keith y Steven Brain
Groeswen

Capítulo I

PRIMEROS CONTACTOS CON LA PROGRAMACION

La decisión de empezar a escribir sus propios programas de juegos es un problema un poco parecido al del huevo y la gallina. Sin embargo, recordando que la mayoría de los juegos de acción se basan en unas reacciones rápidas, empezaremos por contemplar la idea —muy sencilla— de preparar un comprobador de reacciones que mida la rapidez con que usted es capaz de responder (pulsando una tecla) a la aparición de un carácter en la pantalla.

UN COMPROBADOR DE REACCIONES

Lo primero que hay que hacer es borrar la pantalla (Diagrama de flujo 1.1).

```
10 CLS
```

La forma más sencilla de medir el tiempo con un Dragón es utilizando la función TIMER del BASIC, función que accede al reloj interno. Para empezar a medir el tiempo, inicializamos TIMER a cero. Luego, podemos imprimir nuestro carácter.

```
70 TIMER=0
80 PRINT"*"
```

Ahora tenemos que comprobar si se ha pulsado alguna tecla y, si esto no ha ocurrido (en otras palabras, si A$ es la cadena vacía), repetir y comprobar otra vez.

```
100 A$=INKEY$
110 IF A$="" THEN 100
```

Cuando finalmente se apriete cualquier tecla, imprimiremos el tiempo de reacción inmediatamente debajo del carácter (puesto que no hay ninguna puntuación al final de la línea 80).

```
150 PRINT TIMER
```

Resulta muy sencillo conseguir tiempos de reacción buenos con este programa corto ya que el carácter se imprime inmediatamente (y usted, probablemente, tiene todavía su dedo sobre ENTER). Para hacer las cosas un poco más difíciles necesitamos introducir un retardo aleatorio de tiempo antes de la aparición de cada carácter (Diagrama de flujo 1.2) de forma que se tenga realmente que estar atento y esperar antes de pulsar una tecla.

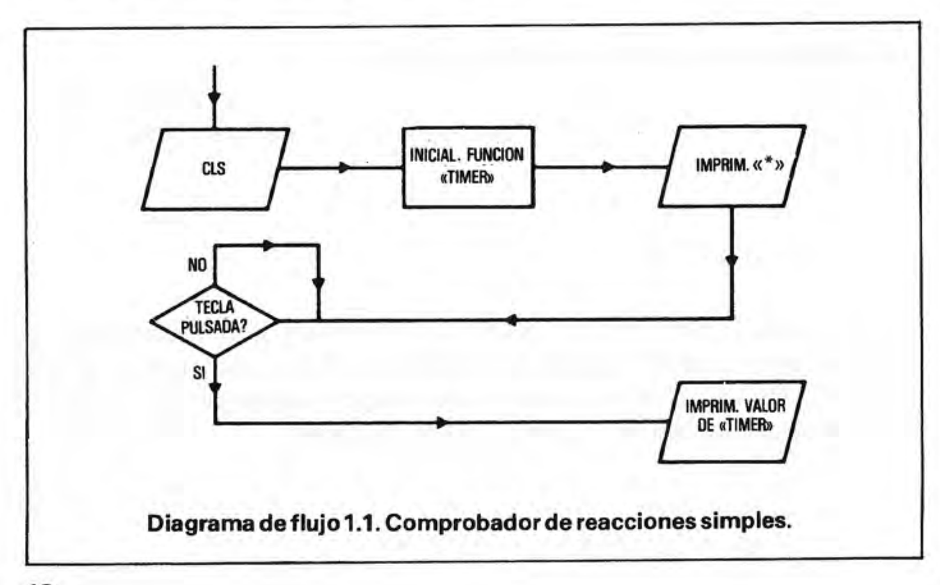


Diagrama de flujo 1.1. Comprobador de reacciones simples.

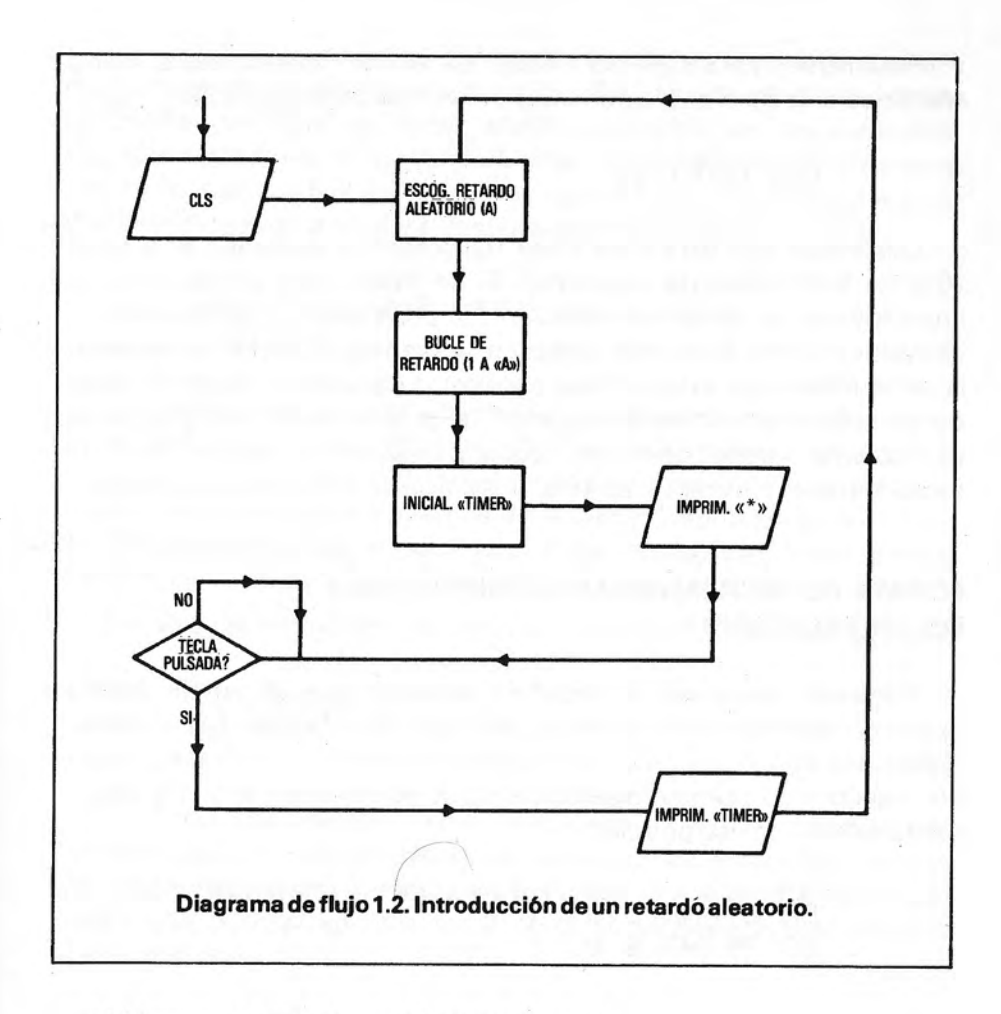


Diagrama de flujo 1.2. Introducción de un retardo aleatorio.

Primero escogemos un número aleatorio.

```
20 A=RND(500)
```

y luego utilizamos este número para fijar el número de ciclos que debe recorrer el bucle de retardo FOR-NEXT.

```
30 FOR N=1 TO A
40 NEXT N
```

Finalmente, para evitar el trabajo de entrar repetidamente RUN, añadimos una línea que automáticamente reinicie la ejecución.

```
160 GOTO 20
```

Obsérvese que esta línea hace retroceder la ejecución a la línea 20 y no al principio del programa. Si se hiciera esto último, el CLS impediría leer su tiempo de reacción. En cada intento, se imprime sucesivamente una línea más abajo en la pantalla. Cuando se alcance la parte inferior de ésta, la línea superior desaparecerá disponiéndose así de nuevo espacio para imprimir. Tal y como está, este programa es bastante bueno pero, en seguida, resulta demasiado fácil: la estrella aparece siempre en una posición perfectamente predecible.

FORMA DE INCREMENTAR LA DIFICULTAD DE UN PROGRAMA

Podemos aumentar la dificultad haciendo que la estrella pueda aparecer aleatoriamente en cualquier posición. Existen 512 posibles posiciones donde imprimir, numeradas del 0 al 511. Por tanto, basta con escoger un número aleatorio situado entre el 0 y el 511 e imprimir la estrella en esa posición.

```
50 B=RND(512)-1
80 PRINT @ B,"*"
```

Obsérvese que, puesto que RND(X) (con X mayor que 1), da un valor entre 1 y X, debemos emplear RND(512) y luego restar 1 si debemos obtener números comprendidos entre el 0 y el 511.

Ahora estamos consiguiendo que la estrella se imprima en posiciones aleatorias pero, al mismo tiempo, hacemos que el tiempo de reacción también aparezca en cualquier lugar. Adecentemos un poco las cosas y hagamos que el tiempo de reacción aparezca siempre en la esquina superior izquierda de la pantalla.

```
150 PRINT @ 0,"TIEMPO";TIMER
```

Las cosas aparecen más ordenadas ahora pero, como la pantalla se va llenando, el tiempo de reacción del jugador aumenta porque no puede distinguir la última estrella aparecida de las anteriores. Hay dos soluciones para este problema. Se puede generar un sonido con la instrucción SOUND cada vez que aparezca una nueva estrella o podemos borrar la pantalla después de cada intento.

```
80 PRINT @ B,"*":SOUND 255,1
140 CLS
```

Obsérvese que debemos borrar la pantalla justo antes de que se imprima el tiempo pues, de otra forma, no podríamos verlo nunca.

Incluso así, no tarda mucho uno en acostumbrarse al juego. Vamos a requerir del jugador un poco más de habilidad imprimiendo un 1 ó un 2 y obligándole a que presione una tecla (cualquiera) únicamente si aparece un 2. (Diagrama de flujo 1.3).

Primero debemos escoger aleatoriamente un 1 ó un 2 e imprimirlo.

```
70 C=RND(2)
80 PRINT @ B,C:SOUND 255,1
```

En el ejemplo anterior —el de la estrella— sólo había una acción correcta: pulsar cualquier tecla. Sin embargo, en esta nueva versión del juego, las cosas son más complicadas. Para acertar, hay que pulsar una tecla si aparece un 2 pero no se debe pulsar ninguna si aparece un 1.

Pantalla	¿Tecla pulsada?	
2	sí	CORRECTO
2	no	INCORRECTO
1	sí	INCORRECTO
1	no	CORRECTO

Si no se pulsa ninguna tecla en absoluto, el programa entrará en un bucle que irá eternamente de la línea 110 a la 100. Podemos resolver este problema colocando un bucle FOR-NEXT de longitud fija cerca de la comprobación de INKEY$ y volviendo al principio

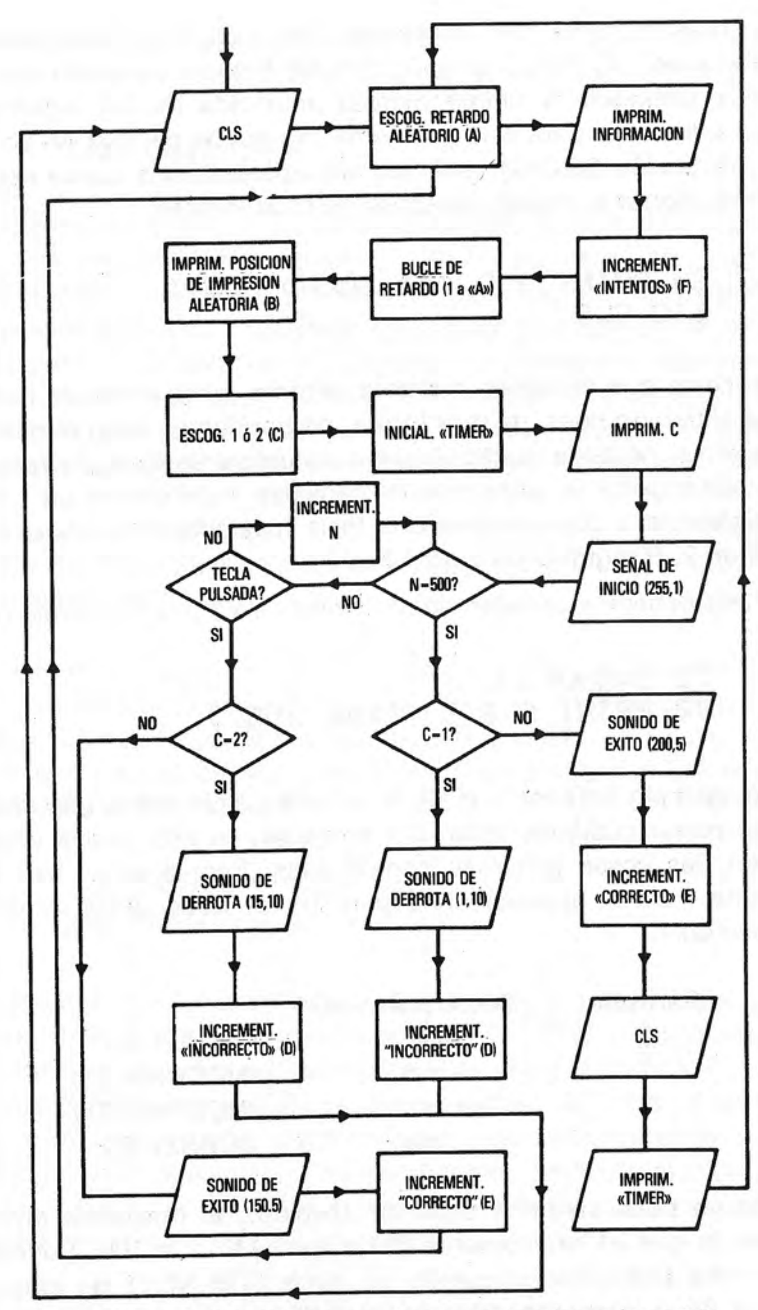


Diagrama de flujo 1.3. Comprobador de reacciones complejo.

(sin imprimir el tiempo si el usuario agota el tiempo disponible o si pulsa una tecla cuando aparezca un 1.)

Comprobemos 500 veces.

```
90 FOR N=1 TO 500
```

Si no hay ninguna tecla pulsada volvemos a comprobar. Si ya se ha comprobado 500 veces, volvemos al principio.

```
110 IF A$="" THEN NEXT N:GOTO 10
```

Si hay una tecla pulsada y se había impreso un 1, volvemos al principio.

```
120 IF C=1 THEN 10
```

LA INCORPORACION DE SONIDO A UN PROGRAMA

Esto empieza a hacerse un tanto complicado así que es mejor que resigamos el diagrama de flujo 1.3 si queremos saber a dónde va antes de continuar.

Para asegurarse de que el lector se da cuenta de sus errores ¿por qué no asociar sonidos a los comportamientos correctos e incorrectos?

Si no se pulsa ninguna tecla, volvemos a comprobar hasta que el tiempo se termine. Si acaba el tiempo y se imprimió un 2 pero no se pulsó ninguna tecla, entonces que suene el sonido de la derrota. Si se acaba el tiempo y no se imprimió un 2, entonces que suene el sonido del éxito.

```
110 IF A$="" THEN NEXT N:IF C=2
THEN SOUND 15,10:GOTO 10:ELSE SO
UND 150,5:GOTO 10
```

Si se pulsa una tecla antes de que termine el tiempo y se imprimió un 1, entonces que suene el sonido de la derrota.

```
120 IF C=1 THEN SOUND 1,10:GOTO
10
```

¡Qué suene el sonido del éxito! Este punto sólo se alcanza si se había pulsado una tecla antes de que terminara el tiempo y había un 2 en la pantalla. Obviamente, no tiene ningún sentido seguir imprimiendo el tiempo si ya se ha terminado el disponible pues siempre se imprimirá lo mismo.

```
130 SOUND 200,5
```

EL CALCULO DE PUNTUACIONES POR ORDENADOR

La vida nos resultaría un poco más sencilla si el ordenador calculara nuestra puntuación por nosotros. Añadamos, pues, tres nuevas variables (D,E y F) para representar respectivamente fallos, éxitos e intentos realizados. Las podemos incluir en las líneas del programa al igual que hicimos con los distintos sonidos. Las incrementaremos apropiadamente después de cada intento.

```
110 IF A$="" THEN NEXT N:IF C=2
THEN SOUND 15,10:D=D+1: GOTO 10:
ELSE SOUND 150,5:E=E+1:GOTO 10
120 IF C=1 THEN SOUND 1,10:D=D+1
:GOTO 10
130 SOUND 200,5:E=E+1
```

Necesitamos imprimir después de cada intento los diferentes resultados en la parte superior de la pantalla. Obsérvese el cuidado que hemos tenido para comprimir toda esta información en la línea superior.

```
20 A=RND(500):PRINT @ 8,"INTENTO
";F;"CORRECTOS";E;"INCORRECTOS";
D:F=F+1
```

La posición aleatoria de impresión se elimina de la línea superior de la pantalla para que no interfiera con la presentación de la información.

```
50 B=RND(480)+32
```

AHORA ES SU TURNO

1) Varíe el tiempo que le hemos dado para reaccionar después de ver aparecer un 1 ó un 2.
2) Haga las modificaciones necesarias para limitar las posiciones de impresión a las seis líneas intermedias de la pantalla.
3) Haga que el juego termine después de diez intentos y que se imprima la puntuación final en forma de porcentaje.
4) Añada una rutina que compruebe si el usuario ha hecho trampa pulsando una tecla antes de que se imprimiera el carácter en la pantalla (Pista: compruebe INKEY$ antes de imprimir el carácter).

Listado completo del comprobador de reacciones

```
5 REM BORRAR PANTALLA
10 CLS
15 REM ESCOGER RETARDO:IMPRIMIR
INFORMACION:INCREMENTAR INTENTOS
20 A=RND(500):PRINT @ 8,"INTENTO
";F;"CORRECTOS";E;"INCORRECTOS";
D;:F=F+1
25 REM BUCLE DE RETARDO
30 FOR N=1 TO A
40 NEXT N
45 REM ESCOGER POSICION IMPRESIO
N
50 B=RND(480)+32
55 REM ESCOGER 1 O 2
60 C=RND(2)
65 REM INICIALIZAR TIMER
70 TIMER=0
75 REM IMPRIMIR NUMERO ESCOGIDO
80 PRINT @ B,C:SOUND 255,1
85 REM COMPROBACION TECLA Y CONS
ECUENCIAS
90 FOR N=1 TO 500
100 A$=INKEY$
110 IF A$="" THEN NEXT N:IF C=2
THEN SOUND 15,10:D=D+1:GOTO 10 E
LSE SOUND 150,5:E=E+1:GOTO 10
```

```
120 IF C=1 THEN SOUND 1,10:D=D+1
:GOTO 10
130 SOUND 200,5:E=E+1
135 REM CLS ANTES DE IMPRIMIR EL
 TIEMPO
140 CLS
145 REM IMPRIMIR TIEMPO DE REACC
ION
150 PRINT @ 0,"TIEMPO";TIMER
155 REM RETURN SIN HACER CLS
160 GOTO 20
```

Capítulo II

LA PREPARACION DE UN JUEGO REAL

En el primer capítulo estudiamos la comprobación de reacciones y vimos cómo una idea muy sencilla podía complicarse mediante la introducción de factores aleatorios y la presentación en pantalla de las puntuaciones obtenidas. A pesar de esta complicación, acabábamos el capítulo distinguiendo entre unos y doses lo cual no es precisamente tarea que requiera una gran capacidad mental. Normalmente los mejores juegos utilizan gráficos y se basan en un buen argumento. Por tanto, ¿qué tal si nos dedicamos a convertir ese esquema inicial de programa en un sencillo juego bélico?

LA INCORPORACION DE GRAFICOS SENCILLOS

Empecemos por pensar en cómo dibujar un gráfico de baja resolución que represente la figura de un hombre. Utilizaremos los caracteres (128) a (143) que consisten en bloques negros y verdes en distintas disposiciones. Cuando se emplean gráficos de baja resolución es necesario que el fondo de la pantalla sea negro. Por tanto, hemos de utilizar CLS0 (negro) en vez de la instrucción por defecto CLS (verde).

La mejor forma de diseñar la figura que se desee es utilizando papel cuadriculado (papel normal para gráficos u hojas para trazadores). Para crear el diseño rellénese los cuadrados y luego divídase en bloques de 2 × 2 cuadrados.

El sistema de gráficos de baja resolución del Dragón crea los gráficos dividiendo cada posición de impresión en cuatro partes de las cuales sólo se activa una. Se puede saber a qué carácter

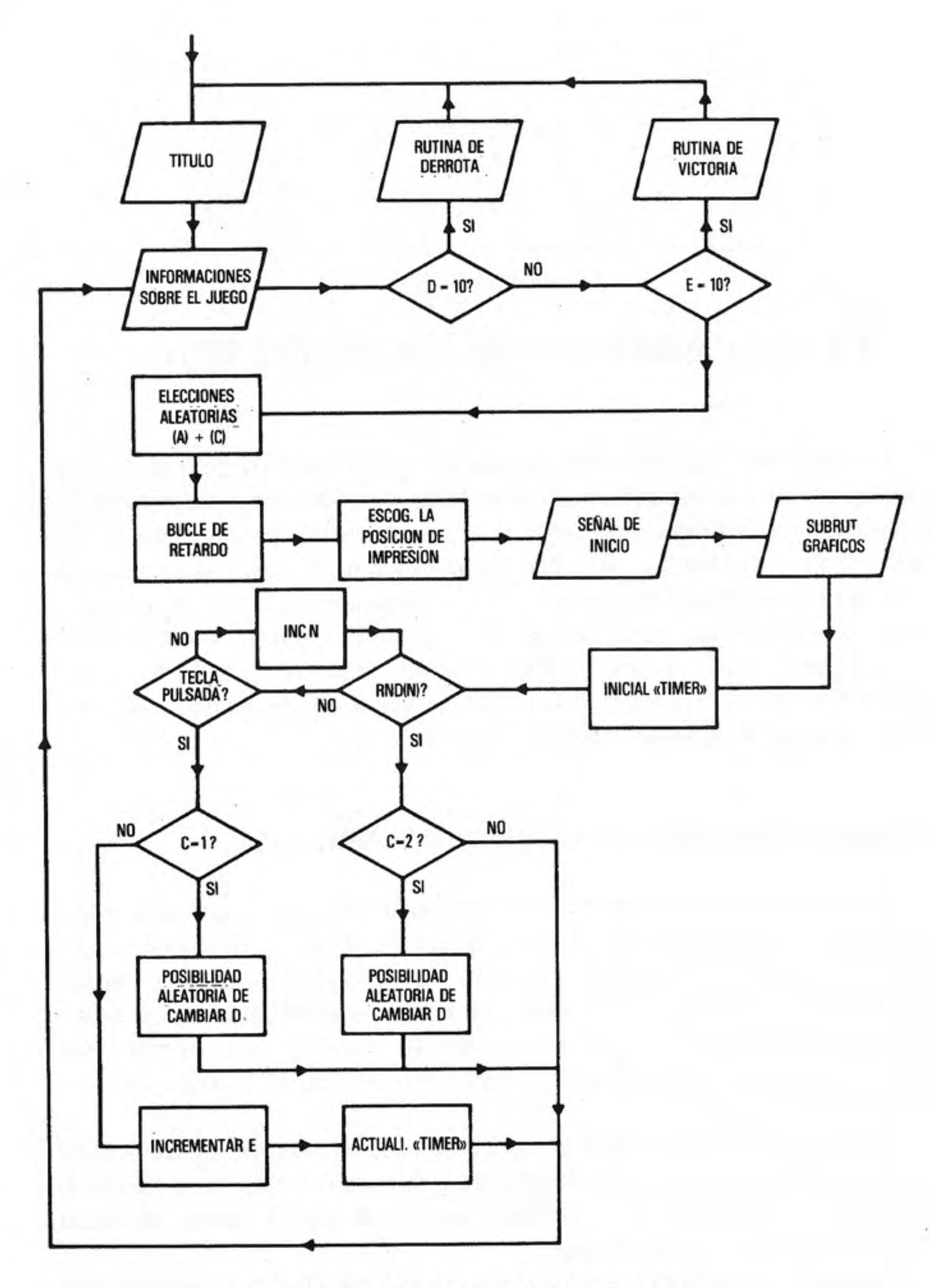


Diagrama de flujo 2.1. «Línea de fuego».

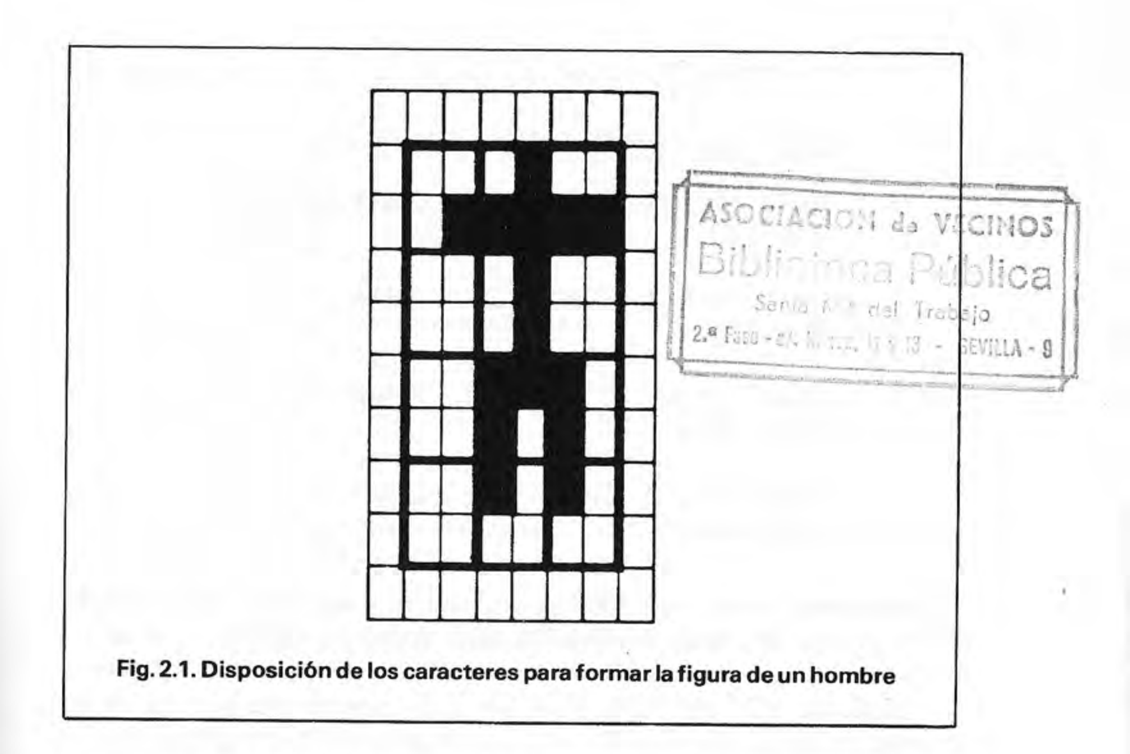


Fig. 2.1. Disposición de los caracteres para formar la figura de un hombre

corresponde cada uno de estos bloques gráficos recurriendo al Apéndice A del manual del Dragón.

Para nuestro hombrecito particular, la cuadrícula tiene que ser de 3 × 4 caracteres y debe contener los siguientes números (Figura 2.1)

129	135	129
128	133	128
128	142	138
128	136	136

Desde luego a los caracteres gráficos del Dragón no se puede acceder directamente desde el teclado. Debe accederse a ellos a través de la función CHR$. Para evitarnos cambiar nuestro programa original más allá de lo estrictamente necesario, pondremos la parte nueva —la que se refiere a los gráficos— en una subrutina que empezará en la línea 1000. El método más sencillo para obtener la figura consiste en imprimir secuencialmente los caracteres que se hayan definido en la cuadrícula.

```
30 CLS 0

1000 REM SUBRUTINA GRAFICA

1020 PRINT CHR$(129);CHR$(135);C
HR$(131);

1030 PRINT CHR$(128);CHR$(133);C
HR$(128);

1040 PRINT CHR$(128);CHR$(142);C
HR$(138);

1050 PRINT CHR$(129);CHR$(136);C
HR$(136);
```

Obsérvese que la línea 1000 es provisional y que luego se escribirá allí otra cosa. Es necesario incluirla si se desea comprobar el programa a medida que se desarrolla ya que la subrutina se llamará siempre mediante GOSUB 1000. Para ver qué aspecto tiene la figura y comprobar que se ha entrado todo correctamente, hágase RUN 1000. El hombre aparecerá en la esquina superior izquierda.

En la práctica, desearemos póder imprimir la @ de nuestro hombre en puntos específicos de la pantalla. Para situar la parte superior de nuestro hombre en una determinada posición (B) podemos emplear PRINT @tal y como ya hemos hecho antes.

```
1020 PRINT @ B,CHR$(129);CHR$(13
5);CHR$(131);
```

La segunda fila de caracteres tiene que estar una línea (32 caracteres) por debajo de ahí aunque, como el primer carácter es el 128, resulta más sencillo dejar esto fuera y desplazar la posición de impresión un carácter (dado que CHR$(128) es el mismo que el color del fondo que hemos escogido mediante CLS0).

```
1030 PRINT @ (B+33),CHR$(133);
```

Análogamente las dos últimas filas se consiguen con:

```
1040 PRINT @ (B+65),CHR$(142);CH
R$(138);
1050 PRINT @ (B+97),CHR$(136);CH
R$(136);
```

Puede comprobarse con RUN 1000. El hombre debe aparecer nuevamente en la esquina superior izquierda ya que el valor por defecto para B (el valor que se obtiene si B no se define en el programa es 0. Inténtese entrar B = 10; GOTO 1000 como un comando directo y obsérvense los movimientos resultantes en la posición de la figura.

Finalmente no debemos olvidarnos de colocar la instrucción RETURN al final de la línea 1050 para retornar de la subrutina.

```
1050 PRINT @ (B+97),CHR$(136);CH
R$(136);:RETURN
```

Para saber en qué lugar de la pantalla puede aparecer la figura hay que dedicar un poco más de atención, sobre todo si no se desea que, por efecto de simetría en la pantalla, la figura se deshaga en pedazos al traspasar los bordes. Introdúzcase B = 28;GOTO 1000 para ver el efecto que acabamos de describir. Restringiremos los movimientos de nuestro hombre a las cercanías del centro de la pantalla y al sentido horizontal. Para evitar el efecto de simetría en los extremos, añadiremos la instrucción B = RND(30) + 255 en la línea 60. Esta instrucción obliga a que la figura aparezca en una de las treinta posiciones de impresión que siguen a la 255 que está en la línea 9.

```
60 B=RND(30)+255:GOSUB 1000
```

LA CREACION DE FIGURAS DE DISTINTO COLOR

Uno de los mayores problemas que se presentan en combate es reconocer rápidamente al enemigo y abatirlo sin volar la cabeza a nuestros propios combatientes. Por tanto, sustituiremos los números

1 y 2, que empleábamos antes, por figuras de dos colores distintos: las verdes son nuestros aliados y el peligro amarillo nuestro enemigo.

Si se consultan los códigos de carácter para los gráficos de baja resolución, se verá que los bloques negros y amarillos necesarios para crear la figura amarilla tienen un código 16 unidades mayor que los correspondientes a los bloques verde y negro. Consecuentemente, para crear un hombre amarillo todo lo que tenemos que hacer es añadir el número 16 a cada una de las funciones CHR$ cuando C = 2. Hay muchas formas de hacer esto pero la más sencilla consiste en sumar una variable (G) a los códigos CHR$ de las líneas 1010 a 1040 y darle el valor cero si C = 1 ó 16 si C = 2 (en la línea 1000 que también da la señal de inicio).

```
1000 SOUND 255,1:IF C=2 THEN G=1
6 ELSE G=0
1020 PRINT @ B,CHR$(129+G);CHR$(
135+G);CHR$(131+G);
1030 PRINT @ (B+33),CHR$(133+G);
1040 PRINT @ (B+65),CHR$(142+G);
CHR$(138+G);
1050 PRINT @ (B+97),CHR$(136+G);
CHR$(136+G); :RETURN
```

EL CAMBIO DEL COLOR DE LAS FIGURAS

Una de las mayores ventajas de este método es que se puede cambiar fácilmente el color de la figura por otro de entre los varios posibles mediante la inclusión de más instrucciones IF-THEN en la línea 1000.

Como antes, las variables D y E se encargan de registrar sus éxitos y fracasos. El tiempo de reacción medio (T) se actualiza ahora en cada ciclo sumando el nuevo tiempo de reacción a T y dividiendo por 2.

```
100 SOUND 200,5:E=E+1:T=(T+TIMER
)/2:GOTO 30
```

Hay que introducir una pequeña modificación para que no sólo se acierte al pulsar una tecla cuando aparezca un hombre amarillo

sino que, cuando dispare contra uno de sus propios hombres o cuando no acierte a un enemigo, exista la posibilidad aleatoria de que el jugador pierda a uno de sus hombres.

Para esto necesitamos añadir aleatoriamente un 0 o un 1 a la puntuación. Ahora bien, esto no puede hacerse con RND(1) ya que siempre da 1. Por otra parte si se emplea RND(0) se obtiene un número entre el 0 y el 1. No existe un método sencillo para convertir RND(0) en 0 ó 1 ya que INT redondearía a cero. La solución más sencilla para obtener un 0 ó un 1 es utilizar RND(2) y restarle 1.

```
80 IF A$="" THEN NEXT N:IF C=2 T
HEN SOUND 15,10:D=D+(RND(2)-1):G
OTO 50:ELSE SOUND 150,5:GOTO 30
90 IF C=1 THEN SOUND 1,10:D=D+(R
ND(2)-1):GOTO 30
```

Con el fin de aumentar la dificultad del juego, podemos hacer que el tiempo durante el que la figura aparezca en la pantalla sea menor y, además, variable. Para ello, podemos hacer que el bucle FOR-NEXT que contiene a la instrucción INKEY$ sea de longitud aleatoria.

```
70 TIMER=0:FOR N=1 TO RND(50)+50
:A$=INKEY$
```

LA CREACION DE HISTOGRAMAS

En vez de contentarnos con imprimir cifras para mostrar la puntuación obtenida, podríamos añadir a la línea 30 (que actualiza la pantalla) una presentación gráfica de barras (histograma) de las distintas informaciones referidas al juego. Imprimiremos un bloque gráfico verde (CHR$(133)) por cada uno de nuestros hombres y uno amarillo por cada uno de nuestros enemigos (CHR$(149)) (Figura 2.2). Se utilizan bloques verticales de media anchura para indicar el número de enemigos que aún quedan. Para imprimir el número correcto de enemigos se puede utilizar un bucle FOR-NEXT de longitud E (número de aciertos). Este número está situado en la segunda línea de la pantalla (a partir de la posición 32). Para nuestros hombres

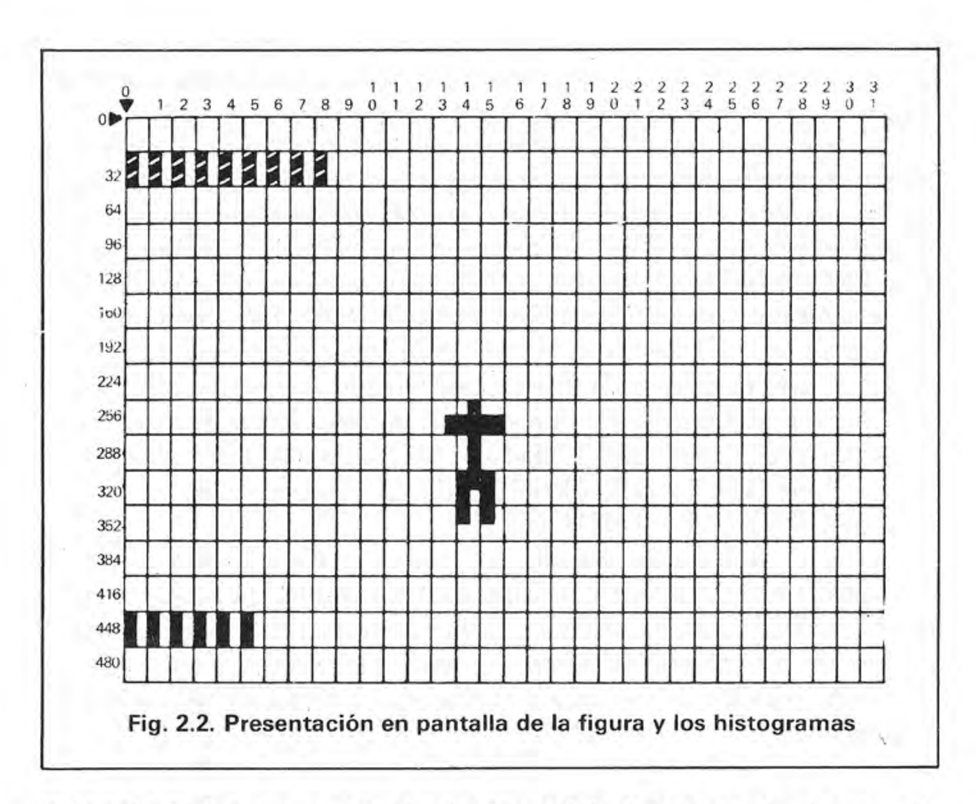


Fig. 2.2. Presentación en pantalla de la figura y los histogramas

la longitud tiene que ser el número original menos el número de fallos (D). Esta otra barra aparece en la parte inferior de la pantalla.

```
30 CLS0:FOR N=0 TO E:PRINT @ (32
+N),CHR$(149);:NEXT N:FOR N=1 TO
(10-D):PRINT @ (448+N),CHR$(133)
;:NEXT N
```

El juego termina cuando se han abatido todos los enemigos (E=10) o cuando todos nuestros hombres han muerto (D=10). El final de los enemigos puede detectarse comprobando el valor de E. La desaparición de nuestros soldados se detecta de forma similar a partir del valor de la variable D.

```
30 CLS0:FOR N=0 TO E:PRINT @ (32
+N),CHR$(149);:NEXT N:FOR N=1 TO
```

```
(10-D):PRINT @ (448+N),CHR$(133)
;:NEXT N:IF D=10 THEN 2000 ELSE
IF E=10 THEN 3000
```

A medida que aumente la destreza del lector podrá utilizar números mayores de amigos y enemigos. Hay espacio suficiente para presentar en pantalla más de 100 enemigos muertos.

SUBRUTINAS PARA LA CONTABILIDAD DE UN PROGRAMA

Ahora que ya tenemos un juego real, debemos añadir algunas subrutinas que nos informen acerca de las partidas que llevamos ganadas y de las que llevamos perdidas. Estas subrutinas se activarán cuando D o E sean iguales a 10. Darán el estado final de las fuerzas en combate y el tiempo medio de reacción positiva. Obsérvese que se ha mejorado un poco la presentación para que el texto aparezca sobre un fondo de color. Se ha conseguido esto utilizando la instrucción PRINT @ para posiciones determinadas y terminando cada frase con un punto y coma. La evaluación de su tiempo de reacción se guarda con una precisión de dos cifras decimales gracias a la instrucción PRINT USING « # # # . # # »;T Se acompaña esta instrucción con el sonido generado al utilizar como argumento de la instrucción SOUND el número empleado en la instrucción PRINT USING.

LA MISION DE LOS EFECTOS DE SONIDO

En caso de que el jugador gane la partida, oirá un sonido ascendente que habrá sido generado por el bucle FOR-NEXT que contiene la instrucción SOUND N,1. Cuando pierda, podrá escuchar un sonido descendente obtenido con otro bucle FOR-NEXT. Tras este sonido, el programa empieza de nuevo. El sonido que indica cuántos hombres ha perdido usted utiliza la variable D + 1 puesto que si se diera el caso de que D fuera igual a cero (lo cual significaría que el jugador es excepcional) se produciría un error: el cero no produce sonidos correctos.

```
2000 CLS4:PRINT @ 41,"ACABAS DE
PERDER";:PRINT @ 71,"A TU ULTIMO
 COMPA/ERO";:SOUND 1,10
2010 PRINT @ 228,"PERO MATASTE";
E;"ENEMIGOS";
2020 PRINT @ 322,"TU TIEMPO MEDI
O DE DISPARO";:PRINT @ 360,"HA S
IDO";:PRINT USING"###.##";T;:SOU
ND(INT(T)),10
2030 FOR N=255 TO 1 STEP-5:SOUND
 N,1:NEXT N:RUN
```

```
3000 CLS2:FOR N=1 TO 255 STEP 5:
SOUND N,1:NEXT N
3010 PRINT @ 35,"FELICIDADES";:P
RINT @ 97,"HAS DESTRUIDO A LA FU
ERZA ENEMIGA";
3020 PRINT @ 226,"HAS PERDIDO";D
;"DE TUS HOMBRES";:SOUND 1,D+1:P
RINT @ 290,"TU TIEMPO MEDIO DE D
ISPARO";:PRINT @ 326,"HA SIDO";:
PRINT USING"###.##";T;:SOUND INT
(T) ,100:RUN
```

Finalmente, conviene recordar que no es mala idea añadir unas cuantas instrucciones para los posibles jugadores así como una secuencia animada de presentación.

```
10 CLS3:PRINT @ 171,"LINEA DE FU
EGO";:PRINT @ 257,"PULSAR CUALQU
IER TECLA PARA DISPARAR A TU ENE
MIGO";:PRINT @ 292,"EL PELIGRO A
MARILLO";:PRINT @ 385,"PERO ASEG
URATE DE NO TOCAR A";:PRINT @ 42
3,"TUS COMPA/EROS LOS VERDES! ";

20 PLAY"T50AGAGAGAGAGAGAGAGAGAGA
GAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGA
GAGAGAGAGAGAG"
```

El comando PLAY de la línea 20 es, simplemente, una forma de ralentizar el programa mientras el jugador digiere las instrucciones de juego y la presentación.

AHORA ES SU TURNO

1) Cambie el color del enemigo por el rojo.
2) Haga que el color del enemigo cambie aleatoriamente de rojo a amarillo, pero asegúrese de que los enemigos sigan siendo fácilmente reconocibles.
3) Titule los distintos histogramas que aparecen en pantalla con las palabras ENEMIGOS Y ALIADOS e imprima la palabra FUEGO! cuando aparezcan los enemigos. Procure que con esta modificación no se altere la presentación.
4) Diseñe otra figura que represente a un enemigo rindiéndose. Cuente este tipo de enemigos como PRISIONEROS.
5) Cambie los sonidos empleados para indicar la derrota o la victoria por verdaderas melodías. Utilice la instrucción PLAY.

Listado completo de «Línea de fuego»

```
5 REM TITULO E INSTRUCCIONES
10 CLS3:PRINT @ 171,"LINEA DE FU
EGO";:PRINT @ 257,"PULSAR CUALQU
IER TECLA PARA DISPARAR A TU ENE
MIGO";:PRINT @ 292,"EL PELIGRO A
MARILLO";:PRINT @ 385,"PERO ASEG
URATE DE NO TOCAR A";:PRINT @ 42
3,"TUS COMPA/EROS LOS VERDES! ";
20 PLAY"T50AGAGAGAGAGAGAGAGAGAGA
GAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGA
GAGAGAGAGAGAG"
25 REM ACTUALIZACION DE PANTALLA
30 CLS0:FOR N=0 TO E:PRINT @ (32
+N),CHR$(149);:NEXT N:FOR N=1 TO
(10-D):PRINT @ (448+N),CHR$(133)
;:NEXT N:IF D=10 THEN 2000 ELSE
IF E=10 THEN 3000
```

```
35 REM RETARDO ALEATORIO Y COLOR
40 A=RND(500):C=RND(2)
45 REM BUCLE DE RETARDO
50 FOR N=1 TO A:NEXT N
55 REM POSICION ALEATORIA DE IMP
RESION:GOSUB GRAFICOS
60 B=RND(30)+255:GOSUB1000
65 REM NUMERO ALEATORIO DE COMPR
OBACIONES DE TECLA
70 TIMER=0:FOR N=1 TO RND(50)+50
:A$=INKEY$
75 REM CONSECUENCIAS
80 IF A$="" THEN NEXT:IF C=2 THE
N SOUND 15,10:D=D+RND(2)-1:GOTO
30:ELSE SOUND 150,5:GOTO 30
90 IF C=1THEN SOUND 1,10:D=D+RND
(2)-1:GOTO 30
100 SOUND 200,5:E=E+1:T=(T+TIMER
)/2:GOTO 30
995 REM SUBRUTINA GRAFICOS
1000 SOUND 255,1:IF C=2 THEN G=1
6 ELSE G=0
1020 PRINT @ B,CHR$(129+G);CHR$(
135+G);CHR$(131+G);
1030 PRINT @ (B+33),CHR$(133+G);
1040 PRINT @ (B+65),CHR$(142+G);
CHR$(138+G);
1050 PRINT @ (B+97),CHR$(136+G);
CHR$(136+G); :RETURN
1995 REM HAS PERDIDO
2000 CLS4:PRINT @ 41,"ACABAS DE
PERDER";:PRINT @ 71,"A TU ULTIMO
 COMPA/ERO";:SOUND 1,10
2010 PRINT @ 228,"PERO MATASTE";
E;"ENEMIGOS";
2020 PRINT @ 322,"TU TIEMPO MEDI
O DE DISPARO";:PRINT @ 360,"HA S
IDO";:PRINT USING"###.##";T;:SOU
ND(INT(T)),10
2030 FOR N=255 TO 1 STEP-5:SOUND
 N,1:NEXT N:RUN
```

```
2995 REM GANAS
3000 CLS2:FOR N=1 TO 255 STEP 5:
SOUND N,1:NEXT N
3010 PRINT @ 35,"FELICIDADES";:P
RINT @ 97,"HAS DESTRUIDO A LA FU
ERZA ENEMIGA";
3020 PRINT @ 226,"HAS PERDIDO";D
;"DE TUS HOMBRES";:SOUND 1,D+1:P
RINT @ 290,"TU PROMEDIO DE PRECI
SION DE TIRO";:PRINT @ 326,"HA T
ERMINADO EL TIEMPO";:PRINT USING
 "###.##";T;:SOUND INT(T),100:RU
N
```

Capítulo III

MECANISMOS PARA REACCIONES PRECISAS

Ahora que ya hemos introducido los gráficos en nuestro juego vamos a pensar en cómo conseguir una reacción más precisa por parte del jugador ante la aparición de la figura, una reacción más precisa que la simple decisión de pulsar una tecla o abstenerse de hacerlo. Vamos a preparar diez blancos distintos. Ahora, comprobaremos si, en el momento oportuno, hemos pulsado la tecla numérica correspondiente al blanco al que deseamos acertar.

Anteriormente, la posición aleatoria de impresión se determinaba mediante B = RND (30) + 255. Con esta instrucción se especificaba una posición situada en uno de los treinta puntos que forman la línea 9. Cambiaremos esto para que sólo existan 10 posiciones diferentes que haremos corresponder con las teclas numéricas 0 a 9.

```
P=RND(10)-1
```

Para distribuir uniformemente los puntos a través de la pantalla tenemos que multiplicar ese número por 3. De esta forma, quedarán espaciados de tres en tres posiciones de impresión (P = 0 a 27).

```
B=P*3
```

Finalmente, podemos centrar la presentación en pantalla sobre el eje X con un espacio en cada extremo. Para ello añadimos 257.

```
60 P=RND(10)-1:B=(P*3)+257:SOUND
 255,1:GOSUB 1000
```

LA COMPROBACION DE QUE SE PULSA LA TECLA ADECUADA

Lo que vamos a hacer inmediatamente después es cambiar, las rutinas que se encargan de las «consecuencias de pulsar una tecla». El cambio consistirá en comprobar si se ha pulsado la tecla que verdaderamente corresponde a la posición de la figura. Es necesario, por tanto, comparar la posición de impresión (P) con la tecla pulsada (A$). No es posible comparar cadenas y variables normales directamente. O bien deberemos convertir la variable P en una cadena o bien deberemos transformar A$ en una variable normal.

Se puede emplear cualquiera de las dos soluciones siguientes:

```
80 IF A$="" THEN NEXT N: AK=VAL(
A$):IF AK<>P THEN SOUND 15,10:D=
D+(RND(2)-1):GOTO 30:ELSE SOUND
150,5: GOTO 30
or:
80 IF A$="" THEN NEXT N: IF A$<>
STR$(P) THEN SOUND 15,10: D=D+(R
ND(2)-1):GOTO 30:ELSE SOUND 150,
5:GOTO 30
```

Tener que pulsar la tecla exacta y no otra nos hará la vida bastante más difícil. Si realmente se encuentra con dificultades por esta variación del juego inicial añada la siguiente instrucción que le servirá para entrenarse: hace aparecer sobre la figura el número de la tecla que hay que pulsar.

```
1010 PRINT @ (B-32),P;
```

Probablemente necesitará también más tiempo para reaccionar. Piense en la línea 70.

ENCUENTROS INFORMATICOS EN LA SEGUNDA FASE

A estas alturas, el lector ya debe estar empezando a tener la ilusión de que ha mejorado sus reacciones y de que éstas son más

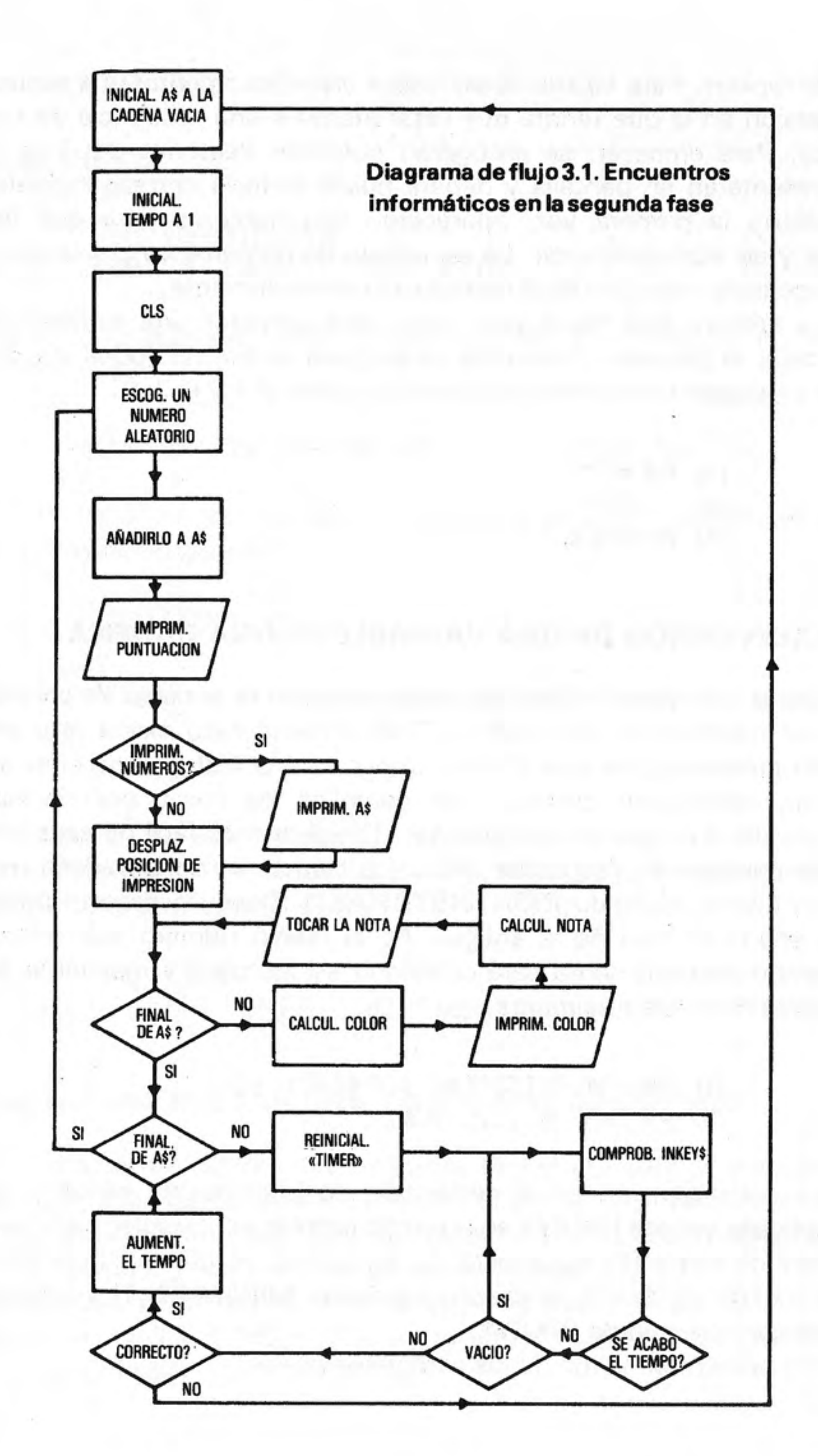


Diagrama de flujo 3.1. Encuentros informáticos en la segunda fase

y más rápidas. Para bajarle de las nubes vamos a preparar una segunda versión en la que tendrá que estar atento a una secuencia de números. Para empezar, se escogerán numeros aleatorios del 1 al 7, se presentarán en pantalla y deberá pulsar la tecla correspondiente. Si acierta la primera vez, aparecerán dos números para que los repita y así sucesivamente. La secuencia de números va creciendo y se hace cada vez más difícil reproducirla correctamente.

Lo primero que habrá que hacer será preparar una cadena vacía, dejar la pantalla totalmente negra para poder ver todos los colores y escoger un número comprendido entre el 1 y el 7.

```
10 A$=""
20 CLS0
30 A=RND(7)
```

LA CONVERSION DE UNA VARIABLE EN UNA CADENA

Como ya hemos visto, hay que convertir la variable A en una cadena mediante la instrucción STR$. En este caso existe otro pequeño problema y es que STR$ siempre da una cadena que empieza por un espacio en blanco. Esto complica las cosas cuando hay que dividir la cadena en subcadenas. La salida más fácil de este problema consiste en despreciar todos los caracteres de la cadena menos el último haciendo RIGHT$(STR$(A),1). Después de esto podemos añadir al final de la antigua A$ el nuevo número que hemos generado aleatoriamente para conseguir así la nueva y más larga A$ que imprimiremos totalmente.

```
40 A$=A$+RIGHT$(STR$(A),1)
70 PRINT @ 256,A$;
```

La comprobación de la corrección de la respuesta es un poco complicada ya que INKEY$ sólo puede aceptar un carácter cada vez. Hemos de tratar los caracteres de la cadena de uno en uno. Para ello, iremos dividiendo la cadena mediante MID$(A$,N,1) y comparando con el siguiente INKEY$.

Del primero al último de los caracteres de A$.

```
160 FOR N=1 TO LEN(A$)
```

Leer INKEY$.

```
180 B$=INKEY$
200 IF B$="" THEN 180
```

Si la tecla pulsada no es la correcta, hay que empezar el programa de nuevo.

```
210 IF B$<>MID$(A$,N,1) THEN 10
```

Si la tecla no es incorrecta, entonces hay que comprobar el siguiente carácter de A$.

```
230 NEXT N
```

Si se ha llegado al final de la cadena, se puede continuar.

```
240 GOTO 20
```

Si esto le parece demasiado fácil, puede añadir un límite de tiempo para pulsar las teclas necesarias. Para este fin utilizaremos el reloj interno mediante TIMER.

```
170 TIMER=0
190 IF TIMER>100 THEN 10
```

TIEMPO PARA PULSAR UNA TECLA: DOS SEGUNDOS

De esta forma, tendrá 100 unidades de tiempo para pulsar cada tecla. Como el temporizador (TIMER) oscila a una frecuencia de 50 Hz, esto supone un tiempo aproximado de dos segundos. También se puede fijar un límite global de tiempo que tenga en cuenta la longitud de la cadena. En esta segunda posibilidad, tendrá que incrementar las líneas 160 y 170 para evitar que el temporizador vuelva a su valor inicial con cada carácter.

```
160 TIMER=0
170 FOR N=1 TO LEN(A$)
190 IF TIMER > (LEN(A$)*30) THEN
 10
```

El tiempo medio para cada carácter es ahora de 30 unidades de tiempo.

Como es natural, cada vez se hace más difícil seguirle la pista a lo que está ocurriendo en la pantalla. Por tanto, vamos a hacer que aparezca la puntuación (que será igual a la longitud de la cadena A$ menos uno).

```
50 PRINT @ 0,"PUNTUACION";(LEN(A
$)-1)
```

LA INCORPORACION DE COLOR A UN PROGRAMA

Resulta un poco aburrido mirar tantos números. ¿Qué le parece si añadimos un poco de color imprimiendo también los bloques gráficos correspondientes a los números? Como antes, podemos dividir la cadena en números individuales (C$).

```
90 FOR M=1 TO LEN(A$)
100 C$=MID$(A$,M,1)
```

Sin embargo, antes de que podamos convertir los números en los correspondientes bloques gráficos de baja resolución debemos transformarlos en variables normales obteniendo el valor de la cadena.

```
110 C=VAL(C$)
```

Para obtener el código del bloque apropiado restamos 1 a C, multiplicamos por 16 el número obtenido y luego le sumamos 143 (recordará del anterior capítulo que los códigos de los bloques de color van de 16 en 16 números a partir del CHR$(143)).

```
120 PRINT CHR$(143+16*(C-1));
150 NEXT M
```

Todo lo que necesitamos hacer ahora es imprimir el bloque debajo del número apropiado. Para ello, desplazamos la posición de impresión después de haber impreso A$.

```
80 PRINT @ 288,"";
```

MUSICA PARA UN PROGRAMA

Si los destellos luminosos no son suficientes para satisfacerle ¿qué le parece un poco de música? En este caso tendremos que convertir los números 1 a 7 en las letras A hasta G para poder introducirlas como parámetros de una instrucción PLAY. Esto no resulta excesivamente complicado. Basta con obtener el código ASCII del número y sumarle 16. Se obtiene así el código ASCII de la letra correspondiente. Si no entiende muy bien cómo funciona este método eche una ojeada a la tabla de códigos ASCII.

```
130 D=ASC(C$)+16
140 PLAY CHR$(D)
```

Para hacer las cosas aún más interesantes podríamos ir cambiando el compás de la música (el *«Tempo»* que dicen los músicos) T a medida que la cadena va aumentando de longitud. Incrementaremos T cada vez que el jugador acierte.

```
220 T=T+1
```

Tiene que inicializar T a 1 en la línea 10 o el programa fallará a la primera puesto que el «Tempo» 0 no es correcto. Análogamente añadiremos en la línea 220 la comprobación de que T no excede nunca el valor 155.

```
10 A$="":T=1
220 T=T+1:IF T>155 THEN T=155
```

Ahora cambiamos la línea 140 para tener en cuenta el valor del «Tempo».

```
140 PLAY"T"+STR$(T)+CHR$(D)
```

LA RUTINA DE PRESENTACION DE UN PROGRAMA

Finalmente añadiremos -para impresionar a los amigos— una rutina que se encargue de presentar el título del programa. Cada vez que se ejecute el programa lo primero que se hará será saltar a esta rutina dónde se guarda una secuencia de notas junto con una presentación de colores y números en movimiento —cosa que nos parece haber oído ya antes en alguna parte.

Los números a imprimir con la melodía inicial y la presentación en color quedan contenidos en N$ y las notas que deben sonar se guardan en P$.

```
1010 N$="1234567"
1020 P$="GDBDG"
```

Durante la presentación inicial del programa se toca la misma secuencia de notas con una duración cada vez mayor (la duración por defecto es 4).

```
1050 PLAY"GDBDL2G"
1070 PLAY"L5GDBDL3G"
1090 PLAY"L6GDBDL4G"
```

Se puede escoger entre sacar por pantalla números y colores o sólo colores (esto queda estrictamente reservado a los expertos).

```
1120 Q$=INKEY$
1220 IF Q$<>"S" AND Q$<>"N" THEN
1120
```

La selección se realiza comprobando con INKEY$ ya que así no se interrumpirá la repetición de la música. Si se pulsa S o N empezará el juego pero se tendrá que añadir una instrucción que salte toda la

sección de impresión de números en caso de que se haya pulsado la tecla N.

```
60 IF Q$="N"THEN 80
```

Se consigue imprimir una línea vertical de números cuando se divide N$ y se imprime en líneas sucesivas añadiendo (32 × N) a la posición inicial de impresión.

```
1170 PRINT @ (334+32*N),MID$(N$,
N,1);
```

Una columna paralela de bloques en color se obtiene de forma análoga con la función CHR$.

```
1190 PRINT CHR$(143+16*(N-1));
```

También se puede producir una nota dividiendo P$ para formar R$. R$ suena seguida de una corta pausa ("P50"):

```
1180 R$=MID$(P$,N,1)
1200 PLAY R$+"P50"
```

La presentación de números y colores se borra cada vez. El borrado se consigue por la sobreimpresión de dos espacios en blanco. Después hay un breve retardo de tiempo. Si INKEY$ no es ni S ni N esta secuencia de operaciones se repetirá indefinidamente.

```
1140 PRINT @ (334+32*N),"  ";
1150 FOR M=1 TO 1000
1160 NEXT M
```

AHORA ES SU TURNO

1) Cambie el programa "Línea de fuego" para que, en vez de pulsar las teclas 0 a 9, haya que pulsar las teclas de la fila inferior (Z a .).

2) Modifique el programa de los «Encuentros Informáticos» para que, después de diez intentos, termine el juego. Haga que al final del juego se muestre la mayor entre las puntuaciones obtenidas (es decir, el mayor número de bloques que haya sido capaz de repetir correctamente).
3) Modifique el programa de los «Encuentros Informáticos» de forma que, después de una respuesta, sólo se borren los dos últimos caracteres de la presentación en vez de tener que volver a empezar desde el principio.

Listado completo de "Encuentros informáticos en la segunda fase"

```
1 GOTO 1000
5 REM INICIALIZACION DE VARIABLE
S
10 A$="":T=1
20 CLS0
25 REM ESCOGER NUMERO
30 A=RND(7)
35 REM A/ADIR NUMERO AL FINAL DE
 A$
40 A$=A$+RIGHT$(STR$(A),1)
45 REM ACTUALIZACION DE LA PUNTU
ACION
50 PRINT @ 0,"PUNTUACION";(LEN(A
$)-1);
55 REM SALTO SOLO PARA COLORES
60 IF Q$="N" THEN 80
65 REM IMPRESION DE NUMEROS
70 PRINT @ 256,A$;
75 REM DESPLAZAMIENTO POSICION D
E IMPRESION
80 PRINT @ 288,"";
85 REM IMPRESION EN COLOR
90 FOR M=1 TO LEN(A$)
100 C$=MID$(A$,M,1)
110 C=VAL(C$)
120 PRINT CHR$(143+16*(C-1));
```

```
125 REM RUTINA DE SONIDO
130 D=ASC(C$)+16
140 PLAY"T"+STR$(T)+CHR$(D)
150 NEXT M
155 REM RUTINA DE COMPROBACION D
E TECLA
160 FOR N=1 TO LEN(A$)
170 TIMER=0
180 B$=INKEY$
190 IF TIMER>100 THEN 10
200 IF B$="" THEN 180
210 IF B$<>MID$(A$,N,1) THEN 10
220 T=T+1:IF T>155 THEN T=155
230 NEXT N
240 GOTO 20
1000 REM RUTINA DE LA SECUENCIA
DE TITULO
1010 N$="1234567"
1020 P$="GDBDG"
1030 CLS0
1040 PRINT @ 38,"ENCUENTROS INFO
RMATICOS";
1050 PLAY"GDBDL2G"
1060 PRINT @ 70,"EN LA SEGUNDA F
ASE";
1070 PLAY"L5GDBDL3G"
1080 PRINT @ 134,"SIGA LA SECUEN
CIA";
1090 PLAY"L6GDBDL4G"
1100 PRINT @ 230,"QUIERE VER";
1110 PRINT @ 288,"NUMEROS Y COLO
RES (S/N)?"
1120 Q$=INKEY$
1130 FOR N=1 TO 5
1140 PRINT @ (334+32*N),"  ";
1150 FOR M=1 TO 1000
1160 NEXT M
1170 PRINT @ (334+32*N),MID$(N$,
N,1);
1180 R$=MID$(P$,N,1)
1190 PRINT CHR$(143+16*(N-1));
```

```
1200 PLAY R$+"P50"
1210 NEXT N
1220 IF Q$<>"S" AND Q$<>"N" THEN
1120
1230 GOTO 10
```



Capítulo IV

LA REALIZACION DE MOVIMIENTOS

A la izquierda y a la derecha del teclado de su ordenador podrá ver los cuatro controles del cursor. Resultan extremadamente útiles en cualquier juego para controlar los movimientos hacia arriba, hacia abajo, a la derecha y a la izquierda. Tan sólo son presentables en pantalla el movimiento hacia arriba (CHR$(126)) y hacia abajo (CHR$(127)). La flecha hacia arriba, de todas formas, puede usarse en una comparación de cadenas como por ejemplo:

```
10 IF A$="↑" THEN..............
```

la flecha hacia la izquierda sólo se puede obtener pulsando SHIFT y la flecha hacia arriba. Los otros controles no se pueden obtener en forma de carácter así que será mejor que olvidemos este método de trabajo.

MOVIMIENTO A TRAVES DE LA PANTALLA

En general, las teclas del cursor sólo se emplean en los programas obteniendo sus valores ASCII a través de INKEY$. Los códigos son distintos según se esté trabajando en letras mayúsculas o minúsculas.

Códigos ASCII para las teclas del cursor

	Tecla SHIFT no pulsada	*Tecla SHIFT pulsada*
Flecha izquierda	8	21
Flecha a derecha	9	93
Flecha hacia arriba	94	95
Flecha hacia abajo	10	91

Para evitar que se produzca una catástrofe si tiene la mala suerte de pulsar SHIFT-0, busque una solución de alta seguridad como puede ser la de comprobar que no se ha pulsado ninguna de las dos teclas.

Para realizar movimientos a través de la pantalla hay que tener en cuenta cómo se ha organizado (o, en argot informático, cómo se ha «mapeado») la pantalla. Una pantalla de texto del Dragón consta de 16 líneas cada una de las cuales tiene 32 caracteres. Cada uno de los 512 caracteres se ha asociado a uno sólo de los números comprendidos entre el 0 y el 511 (en términos matemáticos se ha establecido una correspondencia, se han «mapeado» los caracteres con los números). Este mapeo se ha establecido empezando por la esquina superior izquierda de la pantalla y asociando números secuencialmente. Al principio de cada línea se vuelve a la parte izquierda de la pantalla.

Empecemos en un punto P que esté aproximadamente en medio de la pantalla (posición 238). Pensemos ahora en movernos a la derecha y a la izquierda. Para conseguir la nueva posición de impresión todo lo que tenemos que hacer es restar 1 a P cada vez que se pulsa la tecla de la flecha hacia la izquierda y añadir 1 a P cada vez que se pulsa la tecla de la flecha hacia la derecha.

```
110 P=238
120 PRINT @ P,"*";
140 A$=INKEY$:IF A$="" THEN 140
ELSE A=ASC(A$)
150 IF A=8 OR A=21 THEN P=P-1
160 IF A=9 OR A=93 THEN P=P+1
290 GOTO 120
```

Cuando se pruebe este listado se verá que si va lo suficientemente hacia la izquierda o hacia la derecha pasará respectivamente a la línea superior o inferior.

Recorrer toda una línea para pasar a otra no tiene mucho sentido. Por tanto, convirtamos nuestro asterisco en un tortuga que obedezca directamente nuestras órdenes y se mueva en cualquiera de las cuatro direcciones dejando una estela tras ella. Para subir una línea directamente se resta 32 (la longitud de una línea) a P; para descender, basta con sumar 32 a P.

```
170 IF A=94 OR A=95 THEN P=P-32
180 IF A=10 OR A=91 THEN P=P+32
```

UNA RUTINA PARA EVITAR LAS PULSACIONES ERRONEAS

Aunque, en este caso, no es estrictamente necesario, en general, resulta una buena idea añadir un mecanismo que nos permita —en las rutinas de comprobación de las teclas— rechazar las pulsaciones indeseadas. Esto acostumbra a ser una fuente de errores en programas grandes. Si añadimos ELSE 140 al final de la última comprobación de teclas, el programa retrocederá siempre que se pulse una tecla incorrecta. La rutina completa se muestra en el diagrama de flujo 4.1.

¿No le da una cierta sensación de poder el ver cómo el pobrecito asterisco se ve forzado a moverse hacia donde usted le dirige? Vaya con cuidado: si avanza demasiado, la tortuga caerá víctima de un error FC ya que sólo se admiten posiciones de impresión (valores de P) de 0 a 511. Por tanto, hay que añadir una nueva comprobación que le impida salirse por la parte superior o inferior de la pantalla.

```
190 IF P< 0 THEN P=0
200 IF P>511 THEN P=511
```

PEEK Y LO QUE VIENE DESPUES

La posibilidad de detectar hacia dónde vamos resulta esencial en muchas ocasiones. Hay varios juegos muy sencillos (que acostumbran a recibir nombres relacionados con serpientes u otros pavorosos seres escurridizos) que se basan en el principio de esquivar la estela que el propio jugador va dejando al moverse por la pantalla. Para ver lo que hay en el lugar al que nos dirigimos necesitamos mirar qué hay en esa nueva posición de impresión. Esto se puede hacer mediante la instrucción PEEK aplicada sobre el texto de la pantalla. Como el inicio del texto (la posición de impresión 0) está en la posición de memoria 1024 necesitamos sumar este número a P antes de mirar. Una vez hemos mirado (mediante la instrucción PEEK) en la siguiente posición, podemos comparar el valor de lo que allí encontramos con el valor de un carácter en particular.

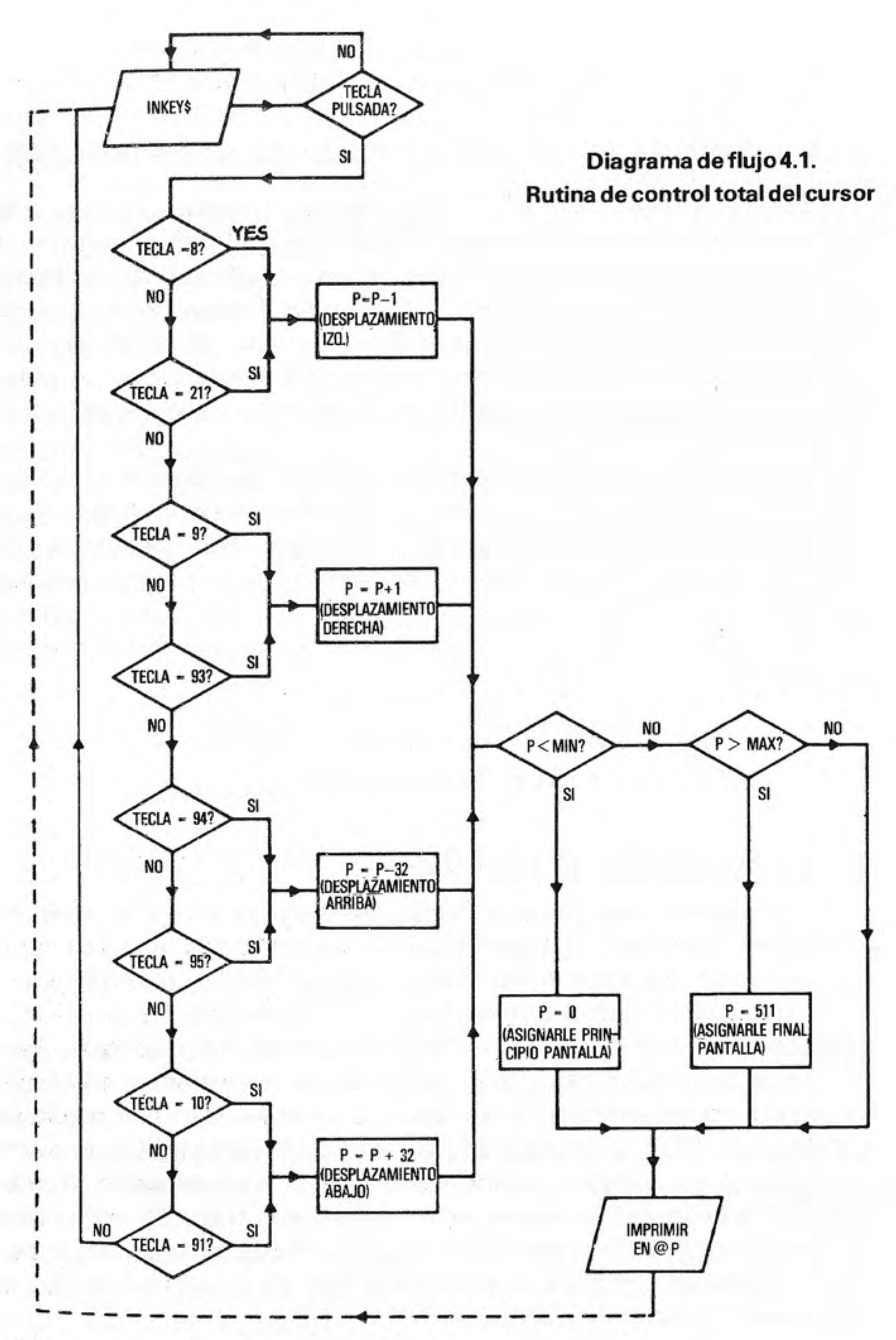


Diagrama de flujo 4.1.
Rutina de control total del cursor

LOS VALORES PEEK Y LOS CODIGOS ASCII PARA LOS CARACTERES DEL DRAGON

Desgraciadamente se descubrirá que si se imprime un carácter en la pantalla y luego se hace PEEK para ver cuál es, se puede recibir una sopresa ya que PEEK puede no dar el valor ASCII correcto del carácter. Para aumentar la confusión, la instrucción PEEK se comporta así tan sólo unas veces mientras que otras no. Esto se debe a la forma en que el Dragón codifica internamente los caracteres. Si se quieren conocer los valores PEEK y los códigos ASCII para el conjunto completo de caracteres del Dragón, se puede emplear la siguiente y breve rutina.

```
500 FOR N=1 TO 255:PRINT CHR$(N)
;:NEXT N
510 FOR N=(1023+1) TO (1023+255)
:PRINT PEEK(N);:NEXT N
```

El valor de PEEK para el asterisco es 106. Por tanto, tenemos que compararlo con él.

```
220 W=PEEK(1024+P)
230 IF W=106 THEN 300
300 PRINT @ 0,"CRASH!"
```

Para saber hasta qué distancia se llegó antes de chocar, añadamos una variable para la distancia (DI) que se incremente tras cada movimiento correcto.

```
270 DI=DI+1
310 PRINT "PUNTUACION=";DI
```

Si quiere moverse sin dejar una estela, necesitará borrar el último punto que ha trazado antes de pasar a imprimir el siguiente. Se puede hacer esto de la forma más sencilla mediante un devastador CLS que borrará toda la pantalla.

```
210 CLS
```

Una solución más elegante sería la que le permitiese actuar de forma más selectiva borrando su última posición pero dejando intacto el resto de la pantalla. Para conseguirlo tiene que recordar la posición de impresión actual e imprimir ahí un espacio en blanco antes de pasar a imprimir un nuevo asterisco. Utilizaremos una nueva variable, LP (última posición), que debe igualarse a P ANTES de cambiar nada.

```
130 LP=P
210 PRINT @ LP," ";
```

Vamos a analizar un ejemplo para asegurarnos de que usted ha entendido bien el funcionamiento de este método. Empezaremos por imprimir "*" @ 238. Si ahora pulsamos la tecla de la flecha hacia la izquierda, P disminuirá a 237 y, como se trata de un número válido, será aceptado. Se imprimirá ahora un espacio en blanco en LP (LP = 238) y luego un nuevo * en P(237).

Debido al orden que siguen las líneas del programa, el asterisco aparece durante la mayor parte del tiempo ya que el borrado sólo se produce cuando se cambia de posición.

LA FORMA DE EVITAR UN «SCROLL» AUTOMATICO

Vamos a reunir todas estas ideas en una situación más interesante (diagrama de flujo 4.2) en la que el «Horror hambriento» debe buscar una caja de comida para no morir de hambre. El horror hambriento no debe cometer el error de devorarse a sí mismo.

Tenemos que dibujar la caja dejando la línea inferior despejada para que no se produzca un desplazamiento vertical automático («Scroll»).

```
10 CLS2
```

Primero, la parte superior:

```
20 PRINT @ 0,STRING$(33,128)
```

luego, los lados:

```
30 FOR N=63 TO 479 STEP 32:PRINT
 @ N,CHR$(128);CHR$(128); NEXT
```

y la parte inferior:

```
40 PRINT @ 449,STRING$(31,128);
```

Ahora colocamos una cantidad aleatoria de comida (Q) en posiciones aleatorias (FP):

```
50 Q=RND(20)
60 FOR N=1 TO Q
70 FP=RND(480)-1
80 PRINT @ FP,":";
90 NEXT N
```

SUBRUTINAS PARA CLARIFICAR CIERTOS GRAFICOS

Si no le gusta ver toda esa comida pegada en la pared, utilice las dos subrutinas al revés. Toda comida colocada en posiciones inadecuadas será borrada mientras se dibujan las paredes. La lógica del ordenador no es siempre un substituto del sentido común.

El monstruo empieza su recorrido con el estómago lleno (FO = 20) y siempre que encuentra un " " (valor PEEK 122) aumenta sus provisiones en 10 unidades.

```
240 IF W=122 THEN FO=FO+10
```

Pero utiliza una unidad de alimento cada vez que realiza un movimiento. El monstruo muere sino le queda comida.

```
260 FO=FO-1
280 IF FO<1 THEN 330
330 CLS0
340 PRINT @ 224,"TE MUERES DE HA
MBRE DESPUES DE";DI;"MOVIMIENTOS
!"
350 SOUND 1,50
360 RUN
```

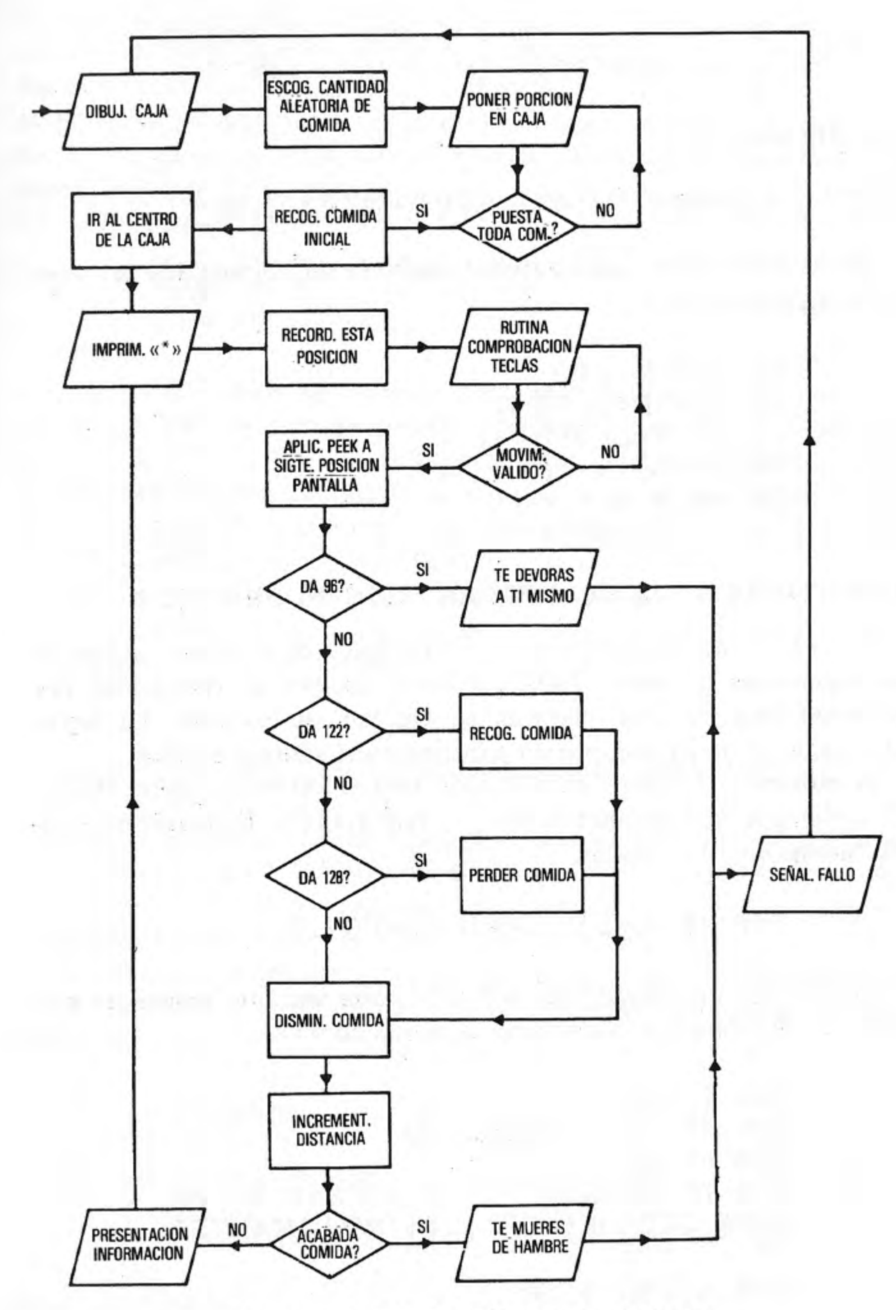


Diagrama de flujo 4.2. «El horror hambriento»

Si choca contra la pared de la caja, pierde cinco unidades.

```
250 IF W=128 THEN FO=FO-5
```

Si mantenemos en el programa la rutina descrita anteriormente, el "horror" deja una estela verde. Esto es así porque anteriormente se borró la pantalla para dejarla de color amarillo mediante la instrucción CLS2 de la línea 10 y la instrucción PRINT dará un bloque verde de valor PEEK 96. Si se quita la línea 210, en cambio, se dejará una estela de asteriscos (valor PEEK 106).

```
240 IF W=96 THEN 300
300 CLS4
310 PRINT @ 224,"TE DEVORAS A TI
MISMO DESPUES DE";DI;"MOVIMIENT
OS"
320 GOTO 350
```

Para darle una idea de cómo lo está haciendo usted, colocaremos en la línea inferior información acerca de la comida y la distancia. Tendremos que añadir para ello algunas cosas a la línea 290. El punto y coma detrás de DI es esencial para impedir el desplazamiento vertical de la pantalla.

```
290 PRINT @ 480,"COMIDA";FO;"DIS
TANCIA";DI;:GOTO120
```

Se observará que este juego se parece mucho a la vida real ya que, se haga lo que se haga, se acaba por morir de hambre cuando se termina la comida.

PROCEDIMIENTOS PARA ADOPTAR DECISIONES MAS RAPIDAS

Tal y como está, el programa no necesita que las tomas de decisiones sean muy rápidas ya que esperará infinitamente a que usted tome una decisión acerca de a dónde dirigirse con el siguiente movimiento. Una manera de obligar al usuario a tomar decisiones veloz-

mente consiste en dejar que continúe el movimiento según la última dirección seleccionada hasta que no se reciba otra orden. Para hacer esto en BASIC necesitamos unos señalizadores que guarden la dirección en la que nos estábamos moviendo. Sin embargo, si piensa un momento en el programa, podrá ver que esos señalizadores ya estaban de hecho allí puesto que A no cambia de valor al hacer A = ASC(A$) hasta que no se escoge una nueva dirección. Todo lo que tenemos que hacer es realizar una mínima alteración al principio de la rutina de comprobación de teclas de forma que, si no se pulsa ninguna tecla, saltemos a la línea 150 en vez de a la 140. La posición de impresión irá cambiando ahora según la última dirección a menos que usted indique expresamente que quiere parar.

```
140 A$=INKEY$:IF A$="" THEN 150
ELSE A=ASC(A$)
```

(si la velocidad es excesiva, se puede reducir introduciendo un bucle de retardo FOR-NEXT en la rutina de comprobación de teclas).

LA FORMA DE VENCER AL SISTEMA

No importa cuánto tiempo se mantenga el dedo sobre una tecla ésta no repetirá su función a menos que no se retire el dedo. Esto se debe a la rutina: antirrebote incluida en el sistema operativo para prevenir repeticiones indeseadas de las acciones de las distintas teclas. Por tanto, si se quiere repetir una acción de forma muy rápida se tendrá que martillear el teclado como un pájaro carpintero a menos que se pueda encontrar un modo de vencer la resistencia del sistema.

Afortunadamente esto no es demasiado difícil aunque requiere utilizar la instrucción PEEK sobre algunas posiciones reservadas de la memoria.

Por ejemplo, un PEEK sobre las posiciones 341 a 344 nos dirá qué tecla de control del cursor se está pulsando ya que la posición correspondiente contendrá entonces el número 223.

PEEKs para el movimiento del cursor

Arriba	341
Abajo	342
Izquierda	343
Derecha	344

Para emplear en el programa del «Horror hambriento» esta técnica que permite la auto-repetición basta con borrar la línea 140 y cambiar las líneas 150-158 para comparar esos PEEKs con el 223.

```
150 IF PEEK(343)=223 THEN P=P-1
160 IF PEEK(344)=223 THEN P=P+1
170 IF PEEK(341)=223 THEN P=P-32
180 IF PEEK(342)=223 THEN P=P+32
```

Se puede trabajar de la misma forma con otras teclas comparando PEEKs específicos con números determinados pero resulta un tanto tedioso. Es mejor una solución que requiera mirar en dos posiciones. Las posiciones más importantes son la 337 y la 135. Si no se pulsa ninguna tecla, la posición 337 contiene el valor 255. Cuando se pulse una tecla, este valor se verá disminuido a otro que dependerá de la tecla pulsada. La posición 135 guarda el código de la última tecla que se ha pulsado. Este valor no se actualiza hasta que no se pulsa una nueva tecla.

Introdúzcase la siguiente línea y luego la instrucción RUN. Se verá cómo el valor guardado en la posición 135 sólo cambia si se pulsa una nueva tecla. Por tanto, puede utilizarse para continuar una acción que hubiera sido anulada.

```
1 PRINT CHR$(PEEK(135)); GOTO 1
```

RUTINA GENERAL DE AUTO-REPETICION

Es fácil escribir una rutina general de auto-repetición que se pare tan pronto como se suelte la tecla. Bastará comprobar si la posición 337 contiene el valor 255 y si no saltar para echar una mirada al código contenido en la posición 135. Si éste se evalúa como una cadena por medio de CHR$ no habrá ningún problema con el rebote inicial de la tecla

ENTER. Este tipo de subrutina se puede emplear como un substituto general de INKEY$ para la auto-repetición.

```
1 ON (PEEK(337)<255)+1 GOTO 1
2 PRINT CHR$(PEEK(135));:GOTO
1
```

LA UTILIZACION DE «JOYSTICKs»

Si en vez de estar encorvado sobre el tablero para jugar, prefiere sentarse cómodamente en su sillón favorito le resultará imprescindible un mando para juegos («joystick»). El mando para juegos contiene dos potenciómetros a los que se accede mediante JOYSTK(0) (horizontal) y JOYSTK(1) (vertical) (ver figura 4.1).

Cada uno dará valores entre 0 y 63 que dependerán de su posición relativa al centro que es aproximadamente 32, 32.

La forma más sencilla de sustituir las teclas del cursor del programa del «Horror hambriento» es cambiar las comprobaciones de los códigos ASCII por otras que comprueben los límites de los movimientos de los mandos.

```
140 J0=JOYSTK(0):J1=JOYSTK(1)
150 IF J0=0 THEN P=P-1
160 IF J0=63 THEN P=P+1
170 IF J1=63 THEN P=P-32
180 IF J1=0 THEN P=P+32
```

Si se deja la palanca en el centro, no pasará nada pero, en cualquier otro caso, si se mueve la palanca hasta su punto máximo de desplazamiento en cada dirección, usted se moverá por la pantalla como antes. Comprobará que el mando para juegos se autorrepite por sí solo. No tiene, por tanto, que preocuparse de este problema.

En la práctica, no resulta una buena idea utilizar los límites máximos de los mandos para juegos ya que algunas unidades tienen pequeños defectos que les impiden llegar al valor 63. Intente colocar en las líneas 150-180 valores distintos para J0 y J1 y observe qué pasa cuando se juega una partida. ¡Fascinante!: acaba de redescubrir el control de la sensibilidad del mando para juegos. Cuanto más cerca del 32 se encuentren

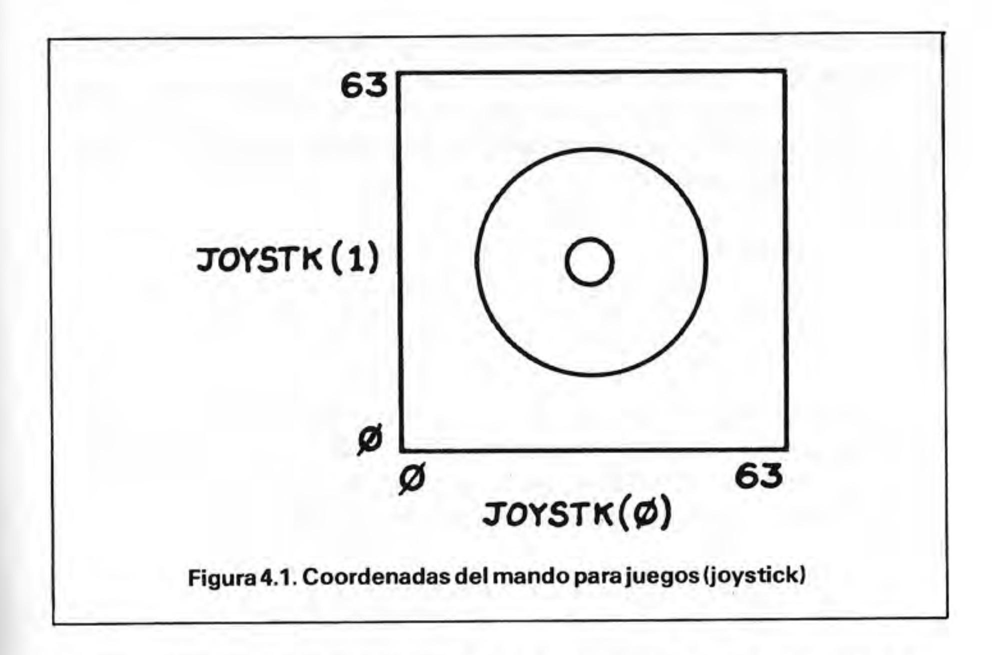


Figura 4.1. Coordenadas del mando para juegos (joystick)

los valores escogidos menor será la distancia que hay que recorrer para obtener una respuesta y, por tanto, más sensible será el control.

También tenemos un botón en el mando de juegos. Aparte de matar marcianos ¿qué podemos hacer con él? ¿Por qué no utilizarlo para recoger comida? Para comprobar si el botón está apretado hay que hacer PEEK(65280) y ver si el valor que hay ahí es el 254. Si se añade esta comprobación a la rutina de detección de comida veremos que sólo nos será posible coger comida si el mando para juegos está siendo pulsado.

```
240 IF W=96 AND PEEK(65280)=254
THEN FO=FO+10
```

El mando para juegos izquierdo puede utilizarse de forma idéntica al mando derecho leyendo JOYSTK(2) y JOYSTK(3) y comprobando si en 65280 hay un 253. Téngase en cuenta que si sólo se va a utilizar el mando derecho, se deberá conectar en la posición derecha. Si se pulsan los botones de ambos mandos habrá un 252 en 65280.

Hasta el momento lo hemos impreso todo en la pantalla pero los mandos para juegos son también muy útiles cuando se utilizan movi-

mientos que requieren gráficos de alta y baja resolución. En estos casos, la pantalla se mapea con coordenadas X e Y que empiezan a contarse a partir de la esquina superior izquierda de la pantalla. Para ver el empleo de JOYSTK en el control de posiciones absolutas introdúzcase la siguiente comprobación.

```
10 CLS0
20 J0=JOYSTK(0):J1=JOYSTK(1)
30 SET(J0,J1,2)
50 GOTO 20
```

Colóquese la palanca del mando en el centro y ejecútese el programa. Probablemente se encontrará un error FC. En baja resolución, la pantalla se mapea de 0 a 63 sobre el eje X pero sólo de 0 a 31 sobre el eje Y. Por este motivo, J1 debe representarse a escala reducida.

```
20 J0=JOYSTK(0):J1=JOYSTK(1)/2
```

A medida que se mueva la palanca se moverá el elemento de la pantalla que se haya seleccionado, dejando una estela amarilla tras de sí. Obsérvese que ahora las posiciones límite de la palanca corresponden a los límites de la pantalla y que si se mueve deprisa se produce una línea de actualización, el elemento de la pantalla parecerá intermitente sin dejar estela alguna.

```
40 RESET(J0,J1)
```

AHORA ES SU TURNO

1) Cambie el programa del «Horror hambriento» de forma que no deje ninguna estela.
2) Escriba un programa que emplee las teclas de control del cursor y en el que un carácter se pueda mover únicamente de una posición par a otra par.
3) Escriba de nuevo el «Horror hambriento» utilizando los gráficos de baja resolución y los mandos para juegos.
4) Añada una instrucción que compruebe si el botón del mando no está permanentemente pulsado (lo que sería una trampa).

Listado completo del «Horror hambriento»

```
10 CLS2
20 PRINT @ 0,STRING$(33,128)
30 FOR N=63 TO 479 STEP 32:PRINT
 @ N,CHR$(128);CHR$(128);:NEXT N
40 PRINT @ 449,STRING$(31,128)
50 Q=RND(20)
60 FOR N=1 TO Q
70 FP=RND(480)-1
80 PRINT @ FP,":";
90 NEXT N
100 FO=20
110 P=238
120 PRINT @ P,"*";
130 LP=P
140 A$=INKEY$:IF A$="" THEN 140
ELSE A=ASC(A$)
150 IF A=8 OR A=21 THEN P=P-1
160 IF A=9 OR A=93 THEN P=P+1
170 IF A=94 OR A=95 THEN P=P-32
180 IF A=10 OR A=91 THEN P=P+32
ELSE 140
190 IF P< 0 THEN P=0
200 IF P>511 THEN P=511
210 PRINT @ LP," ";
220 W=PEEK(1024+P)
230 IF W=96 THEN 300
240 IF W=122 THEN FO=FO+10
250 IF W=128 THEN FO=FO-5:SOUND
1,5
260 FO=FO-1
270 DI=DI+1
280 IF FO<1 THEN 330
290 PRINT @ 480,"COMIDA";FO;"DIS
TANCIA";DI;:GOTO 120
300 CLS4
310 PRINT @ 224,"TE HAS DEVORADO
 A TI MISMO DESPUES DE";DI;"MOVI
MIENTOS!"
```

```
320 GOTO 350
330 CLS0
340 PRINT @ 224,"HAS MUERTO DE H
AMBRE DESPUES DE";DI;"MOVIMIENTO
S!"
350 SOUND 1,50
360 RUN
```

Capítulo V

LOS DESPLAZAMIENTOS VERTICALES EN PANTALLA («SCROLLING»)

En este capítulo nos ocuparemos de cómo crear una representación en pantalla que se desplace verticalmente. Esta presentación contendrá obstáculos que deben esquivarse. También veremos cómo se puede incorporar esta posibilidad a los programas para crear juegos más interesantes. Estudiaremos finalmente cómo se lleva a la realidad una idea compleja. Para ello, reseguiremos todos los pasos necesarios en la preparación de un programa.

La primera cosa a considerar es decidir cuál va a ser el tema del juego. Empezaremos por intentar simular la conducción de un automóvil.

UN PROGRAMA DE ACCION: «A TODA MARCHA»

Primeramente tenemos que ordenar las distintas secciones de que constará el programa y ver en qué orden tenemos que trabajar con ellas. Los siguientes puntos son los más importantes en el desarrollo del programa y están ordenados en la secuencia más sensata.

1. Dibujar la carretera.
2. Colocar obstáculos aleatorios.
3. Colocar nuestro coche en la carretera.
4. Tomar el control de sus movimientos.
5. Comprobar las colisiones que puedan ocurrir.
6. Presentar en pantalla el tiempo transcurrido y la distancia recorrida.
7. Preparar una forma de ganar el juego.
8. Hacer que el programa sea fácil de utilizar.

La primera cuestion a considerar es cómo presentamos en pantalla una tira de asfalto negro. Realmente no se trata de un problema complicado ya que podemos escribir un bloque negro (CHR$(128)) 10 veces.

```
60 PRINT CHR$(128);CHR$(128);CHR
$(128);CHR$(128);CHR$(128);CHR$(
128);CHR$(128);CHR$(128);CHR$(12
8);CHR$(128)
```

Por otra parte, si se piensa un poco, se comprueba que en vez de tener que teclear CHR$(128) diez veces podemos recurrir a la función STRING$ que se encarga de crear una tira de una longitud determinada a partir de un carácter dado. Esta mejora se realiza definiendo la carretera como A$ = STRING$(10,128) e imprimiendo después A$ cada vez que se quiera hacer la misma presentación.

```
20 A$=STRING$(10,128)
60 PRINT A$
```

Si ahora añadimos una línea de retorno y ejecutamos el programa, veremos cómo aparece en la pantalla una carretera negra recta que empieza en la parte izquierda superior y resigue el borde izquierdo hasta llegar a la parte inferior de la pantalla. En ese momento, y gracias al mecanismo automático de «scrolling», (enrollamiento) la impresión de la carretera no se detiene sino que continúa.

```
120 GOTO 20
```

Aunque esto funciona bastante bien tenemos que recordar que las carreteras reales no son así. ¿Qué tal si, para empezar, movemos la carretera hacia el centro de la pantalla? Esto se consigue fácilmente con la instrucción PRINT TAB que desplaza la posición de impresión a una columna determinada.

```
60 PRINT TAB(10);A$
```

Ya tenemos la carretera en el centro de la pantalla pero aún nos encontramos con una carretera demasiado recta, poco real. Por tanto, el paso siguiente será añadir unas cuantas curvas. Inicialicemos la posi-

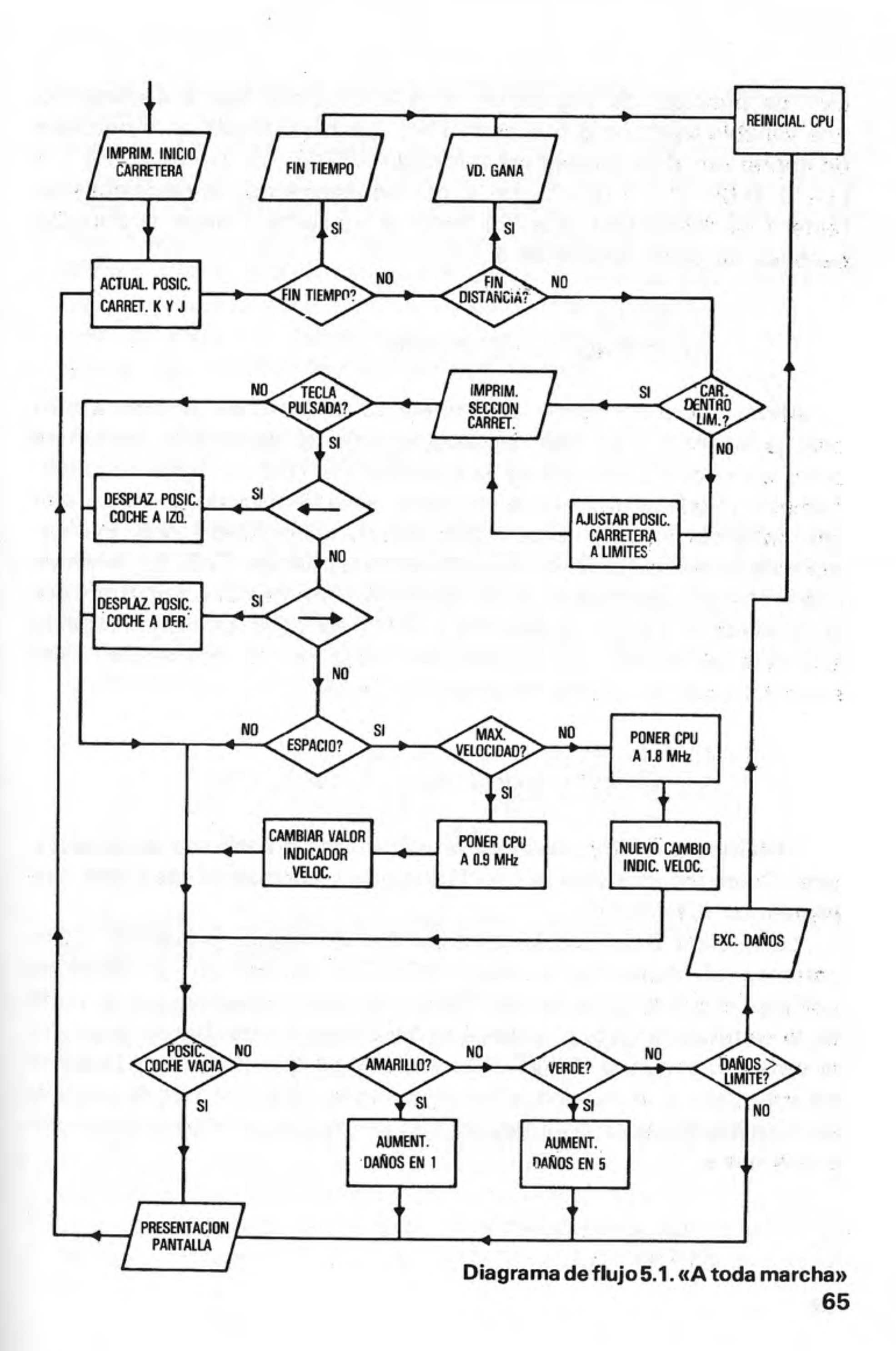


Diagrama de flujo 5.1. «A toda marcha»

ción de principio de impresión (A) a la columna diez y empleemos una variable aleatoria B que se sumará a A para desplazar la posición de impresión. B se conseguirá haciendo RND(3) - 2 que nos dará —1 (1—2), 0 (2—2) o 1 (3—2) con lo que se logrará que la carretera siga recta o se desvíe una posición hacia la izquierda o hacia la derecha después de cada cálculo de B.

```
10 A=10
30 B=RND(3)-2:A=A+B
```

Ahora podrá ver cómo la carretera se desplaza de un lado a otro pero ¡atención!: si se deja desplazarse durante demasiado tiempo se sufre un error FC en cuanto la posición de TAB se haga negativa. También puede ocurrir que la carretera, si va demasiado hacia la derecha, salte a la línea siguiente lo que causará la confusión total. Evidentemente tenemos que limitar los valores de la posición TAB. Lo haremos comprobando continuamente el valor de A. Tomaremos como límite por la izquierda el 1 y por la derecha el 20 (y así dejar suficiente espacio a la derecha de este punto para poder imprimir la carretera en toda su anchura y evitar que se nos salga por el borde).

```
40 IF A>20 THEN A=20
50 IF A<1 THEN A=1
```

Tenemos que colocar ahora unos cuantos obstáculos en la carretera. Como los amarillos se ven fácilmente y además quedan bien emplearemos (CHR$(159)).

La variable C será un número del 0 al 31 (ambos incluidos). Colocaremos los obstáculos mediante PRINT@. No hay que olvidarse de colocar un punto y coma tras CHR$(159) sino se quiere que la parte de la carretera situada a la derecha del obstáculo cambie su color por el verde. El segundo PRINT @ es esencial ya que traslada la posición de impresión a la esquina inferior izquierda de forma que la pantalla se desplace hacia arriba antes de que se imprima la siguiente sección de carretera.

```
110 C=RND(32)-1:PRINT @ (D+480),
CHR$(159);:PRINT @ 511,""
```

Se observará en seguida que los obstáculos no siempre aparecen sobre la propia carretera sino que lo hacen en cualquier punto de la pantalla. La razón de esto reside en que C puede tomar cualquier valor entre el 0 y el 31. Consecuentemente, los obstáculos pueden imprimirse en cualquier columna de la pantalla. Se puede rectificar fácilmente este error si creamos una nueva variable D (un número aleatorio entre 0 y 11) y se la sumamos a A que es la variable que determina en qué lugar de la pantalla sobre una línea horizontal, se debe colocar el número obtenido. Los valores límite de C son ahora de A a A + 10 y así los obstáculos quedarán siempre dentro de la pista.

```
110 D=RND(11)-1:C=D+A:PRINT @ (4
80+C),CHR$(159);:PRINT @ 511,""
```

El paso siguiente es colocar nuestro coche en la carretera. Lo situaremos cerca del centro de la pantalla creando una nueva variable E (de valor inicial 15). La sumaremos a 224 (una posición de impresión en las cercanías del centro de la pantalla) y luego imprimiremos una U en inversa en la posición de impresión que resulte. Con esa U representaremos el coche.

```
10 A=10:E=15
110 D=RND(11)-1:C=D+A:PRINT @ (4
80+C),CHR$(159);:PRINT @ (224+E)
,"u";:PRINT @ 511,""
```

LA ELIMINACIÓN DE LA ESTELA DE LAS FIGURAS AL DESPLAZARSE

Se puede observar que el coche deja tras de sí una estela. Es lógico puesto que no se ha tomado ninguna medida para borrarla. Para poder eliminarla después de cada movimiento tenemos que recordar el anterior valor de E (la última posición del coche), asignárselo a la nueva variable L y luego imprimir un espacio negro en una posición equivalente de la línea superior con lo que, al desplazarse la pantalla hacia arriba, borraremos la última aparición del coche. Obsérvese que el espacio que debemos borrar es el que está en la línea superior a la posición del coche en un momento dado. En realidad el coche se mantiene estático

(ya que lo hemos situado con una instrucción PRINT @) mientras que la pista se mueve con el movimiento de la pantalla.

```
10 A=10:E=15:L=15
110 D=RND(11)-1:C=D+A:PRINT @ (4
80)+C,CHR$(159);:PRINT @ (192+L)
,CHR$(128);:PRINT @ (224+E),"u";
:PRINT @ 511,"":L=E
```

LA UTILIZACION DEL CURSOR

En este momento se debe disponer de una sinuosa carretera negra con obstáculos amarillos así como de un coche que, las más de las veces, está fuera de ella. El paso siguiente permitirá poner el coche bajo el control del jugador mediante las teclas de control del cursor.

Sería perfectamente correcto recurrir a una serie de instrucciones IF-THEN que comprobarán los INKEY$ pero resulta mucho más sencillo y rápido emplear la comprobación lógica de la línea siguiente.

```
70 B$=INKEY$:IF B$="" THEN 100 E
LSE F=ASC(B$):E=E+((F=8)-(F=9))
```

Comprenderemos mejor qué hace esta línea si la dividimos en varias partes:

B$ = INKEY$
(Asignar el valor de INKEY$ a B$)
IF B$ = "" THEN 100
(si no se ha pulsado ninguna tecla saltar a 100)
ELSE F = ASC(B$):
(Si se está pulsando una tecla, entonces darle a F el valor ASCII de esa tecla)
E = E + (F = 8) − (F = 9)
(Disminuir (−1) o aumentar (+1) la posición relativa del coche respecto a la izquierda de la pantalla (E) según se haya pulsado la flecha hacia la izquierda (F = 8) o la flecha hacia la derecha (F = 9). Esto funciona bien ya que (F = 8) y (F = 9) dan TRUE (cierto) y − 1 o FALSE/(falso) y 0, y (0) − (−1) = 1 mientras que (−1) − (0) = −1.

LA FORMA DE DAR DOS VELOCIDADES A LOS JUEGOS

No sólo podemos controlar los movimientos hacia la izquierda y hacia la derecha. También podemos disponer de un cambio de marchas de dos velocidades. Para ello aprovecharemos las posibilidades que ofrece el Dragón para que su CPU trabaje a 0.9 o 1.8 MHz. La velocidad más elevada se obtiene haciendo POKE &HFFD7,0 y la más reducida con POKE&HFFD6,0. Como el Dragón no se diseñó con la intención de que funcionara a 1,8 MHz, es posible que su ordenador no funcione correctamente a la velocidad más rápida debido a problemas con la memoria pero si funciona bien (cosa que, según nuestra experiencia, ocurre la mayor parte de las veces) es un buen medio de dar dos velocidades a los juegos.

También se observará que aumenta la frecuencia de los sonidos que se oyen. Desgraciadamente puede ocurrir que uno no se dé cuenta de que el funcionamiento a velocidad máxima también afecta a las rutinas de carga y almacenamiento en cassette haciendo que sus grabaciones no sirvan para nada (a no ser que previamente se haya pasado a la velocidad más lenta). El programa que estamos desarrollando llega a su fin utilizando la velocidad inferior pero si se detiene con la tecla BREAK se mantendrá el funcionamiento a la velocidad más rápida. Si no se está muy seguro acerca de cuál es la velocidad a la que está funcionando la CPU hágase POKE &HFF6,0 antes de intentar guardar información en el cassette.

Para pasar de una marcha a otra en este programa utilizamos la barra de espaciado que hay en la parte inferior del teclado. Al pulsarla cambiamos el valor de la variable M. M valdrá 0 cuando se vaya a baja velocidad y 1 cuando se circule a la velocidad máxima. Si cuando la CPU funciona a 0,9 MHz se pulsa esa barra (F=32), entonces el valor de M pasa a ser 1 y la velocidad aumenta. En otras palabras: si cuando se circule a baja velocidad, se pulsa la barra, se pasará a velocidad rápida y viceversa.

```
80 IF F=32 AND M=0 THEN POKE &HF
FD7:M=1:GOTO 100

90 IF F=32 AND M=1 THEN POKE &HF
FD6,0:M=0
```

El segundo medio de control que se puede utilizar en este programa es el mando para juegos («joystick»). Se puede emplear la palanca para moverse hacia la izquierda o hacia la derecha y el botón para cambiar de marcha. Si utilizamos este mando tendremos que cambiar la línea de programa que controla la dirección de los movimientos. Su efecto es, sin embargo, similar al de la línea empleada para controlar los movimientos mediante teclas.

```
70 JY=JOYSTK(0):E=E+((JY<10)-(JY
>50))
```

JY es el valor de JOYSTK(0) (el lado derecho del mando respecto al eje derecho) y la E indica lo mismo que cuando usábamos las teclas para controlar la dirección. Si JY es menor que 10 se decrementa E y el coche va hacia la izquierda. Si JY es menor que 50 se incrementa E y el coche va hacia la derecha.

Para pasar a otra marcha cambiese la comprobación que hacíamos para ver si la barra se había pulsado por otra que compruebe el PEEK del botón del mando para juegos.

```
80 IF PEEK(65280)=126 AND M=0 TH
EN POKE &HFFD7,0:M=1:GOTO 100
90 IF PEEK(65280)=126 AND M=1 TH
EN POKE &HFFD6,0:M=0
```

Todo esto está quedando muy bien. Sin embargo, a estas alturas ya debe haber comprobado que puede dirigirse siempre a donde usted quiera. Incluso a las personas que desearían poder circular así les gustaría saber el número de viejecitas que han atropellado, o el de multas y facturas que van a tener que pagar por atravesar en coche jardines ajenos o por llevarse casas por delante.

Una vez más, podemos hacer PEEK sobre la pantalla para ver cuál es nuestra situación. La comprobación más sencilla consiste en SI (IF) el cuadrado al que está a punto de dirigirse el coche no es negro (CHR$ (128)) ENTONCES (THEN) FIN (END).

```
100 IF PEEK(1024+224+E)<>128 THE
N END
```

Aquí aparece un nuevo problema: al principio del juego, antes de que la pantalla a empiece a desplazarse, la carretera no llega aún al centro de la pantalla y el coche queda perdido en medio de un campo. Además, el juego acaba nada más comenzar. Necesitamos, por tanto, imprimir unas cuantas secciones de carreteras extras al principio del programa.

```
20 A$=STRING$(10,128):FOR N=1 TO
 16:PRINT TAB(10);A$:NEXT N
```

SUBRUTINA DE UTILIZACION CONDICIONAL

Ahora, cada vez que usted choca contra un obstáculo o se sale de la pista, termina el juego. Sería más interesante, en vez de acabar tan bruscamente, permitir que el coche siguiera funcionando si bien con daños que se irían acumulando después de cada choque. Lo mejor que se puede hacer es añadir una «variable de daños» (I) a la que sumaremos un 1 cada vez que se choque contra un bloque amarillo (CHR$ (159)) o un 5 cada vez que dé contra un bloque verde al salir de la pista (CHR$(96)). Obsérvese que el verde que se utiliza aquí —dónde no se imprime nada— es el CHR$(96) (un espacio) en vez del bloque gráfico verde que es el CHR$(143). Como las cosas se están haciendo bastante complicadas, es mejor reunir todo esto en una subrutina a la que sólo se accederá cuando el siguiente cuadrado al que se desplace el coche no sea negro.

También hemos añadido unos sonidos de advertencia adecuados. Si el valor de los daños sufridos por el automóvil rebasa la cifra 50, el juego termina.

```
100 IF PEEK(1024+224+E)<>128 THE
N GOSUB 200
200 H=PEEK(1024+224+E)
210 IF H=159 THEN I=I+1:SOUND 10
0,5
220 IF H=96 THEN I=I+5:SOUND 50,
1
230 IF I>=50 THEN 300
240 RETURN
300 END
```

CALCULO DE TIEMPO Y DE DISTANCIAS

Hasta este momento el juego sólo consiste en recorrer una carretera evitando chocar contra unos obstáculos. ¿Por qué no nos planteamos la posibilidad de tener en cuenta el tiempo empleado y la distancia recorrida? El temporizador TIMER se inicializa al valor 0 mediante la instrucción TIMER = 0 y el tiempo en un momento dado se lee mediante la variable K que toma 1/50 del valor de TIMER para dar el tiempo en segundos. La distancia recorrida se incrementa en una unidad en cada vuelta. Deseamos que aparezcan la distancia y el tiempo en la parte superior derecha de la pantalla inmediatamente debajo de las letras D y T. Con este fin se emplea la instrucción de impresión según formato PRINT USING. Por último la instrucción PLAY del final hace sonar una nota muy breve cada vez que se realiza un movimiento. El empleo de la instrucción PLAY en vez de SOUND le permite generar notas más cortas que el valor mínimo aceptado por SOUND que es 1. De esta forma, la ejecución del programa se retrasa menos.

```
20 A$=STRING$(10,128):FOR N=1 TO
 16:PRINT TAB(10);A$:NEXT N:TIME
R=0
30 B=RND(3)-2:A=A+B:K=INT(TIMER/
50):J=J+1
110 D=RND(11)-1:C=D+A:PRINT @ (4
80)+C,CHR$(159);:PRINT @ (192+L)
,CHR$(129);:PRINT @ (224+E),"u";
:PRINT @ 52,"T          D"::PRINT @
53,USING "####";K;:PRINT @ 59,US
ING"####" ;J:: PRINT @511,"":L=E
:PLAY"T255;O4;B"
```

Todavía no le hemos dado una forma de ganar el juego así que vamos a considerar tres factores:

a) Límite de tiempo.
b) Distancia recorrida.
c) Daños sufridos por el coche.

Empecemos por considerar cómo conseguimos perder por agotamiento del límite de tiempo. Todo lo que tenemos que hacer es añadir

una instrucción que compruebe si se ha llegado o no a un determinado límite de tiempo. (1200 en este caso).

```
30 B=RND(3)-2:A=A+B:K=INT(TIMER/
50):J=J+1:IF K=>1200 THEN 400
400 CLS:PRINT"FIN DE TIEMPO. HAS
 RECORRIDO";J;"METROS":GOTO 1000
```

La posibilidad siguiente es ganar por haber recorrido la longitud total de la pista que es de 1.000 metros. (Si parece corta se espere a ver el tamaño del coche).

```
30 B=RND(3)-2:A=A+B:K=INT(TIMER/
50):J=J+1:IF K=>1200 THEN 400 EL
SE IF J=>1000 THEN 500
500 CLS:PRINT"HAS RECORRIDO LOS
MIL METROS EN";J;"SEGUNDOS":GOTO
 1000
```

Los efectos de las colisiones ya se trataron antes pero, si se quiere, se puede añadir también un mensaje.

```
200 CLS:PRINT"TU COCHE SE HA HEC
HO PEDAZOS. HAS RECORRIDO";K;"ME
TROS EN";J;"SEGUNDOS": GOTO 1000
```

AHORA ES SU TURNO

1) Escriba una rutina que le permita escoger entre jugar con el mando para juegos o con las teclas.
2) Aumente la sensibilidad de la dirección cuando emplee el mando para juegos (joystick).
3) Calcule cuántos niveles de dificultad se pueden dar al juego mediante la variación de los distintos factores.
4) Coloque sobre la carretera obstáculos de distintos colores y prepare una rutina que detecte qué tipo de bloque es el que acaba de tocar.
5) Ahora que su programa ya funciona bastante bien puede usted creer

que no hay nada más que hacer. Nada más lejos de la realidad. Ahora le ha llegado el turno a usted y debe aprovecharlo para utilizar su imaginación. Las ideas que se exponen a continuación son sólo sugerencias alternativas que pueden convertirse en nuevas versiones de este juego que está usted empezando a diseñar por sí mismo.

a) Llega tarde al trabajo y debe darse prisa avanzando por una carretera con mucho tráfico.
b) Usted es una rana que acaba de darse cuenta de que no es una buena idea intentar cruzar la carretera.
c) Usted intenta cruzar los tumultuosos rápidos que le separan de un grupo de cazadores situado en la orilla.
d) Usted está intentando atravesar una estrecha abertura en el campo de fuerza del enemigo, esquivando los nódulos de plasma para salir de la Estrella de la Muerte.
e) Es usted una apresurada ama de casa que corre por los pasillos de un multitudinario supermercado, evitando chocar contra los otros clientes y haciendo todo el trabajo que haría su marido si no se hubiera comprado el dichoso ordenadorcito.

La elección es suya. Recuerde que lo que hace a un juego distinto de otro no son sus principios sino la forma en que se ha desarrollado la idea y el nivel de perfeccionamiento al que se ha llegado. Una vez haya decidido sobre qué idea va a desarrollar el juego, debe usted preparar la secuencia de presentación, las instrucciones para el jugador y los mensajes de fin de partida entre otras cosas.

Listado completo de «A toda marcha»

```
5 REM INICIALIZACION DE VARIABLE
S
10 A=10:E=15:L=E
15 REM IMPRESION DE SECCION CARR
ETERA
20 A$=STRING$(10,128):FOR N=1 TO
 16:PRINT TAB(10); A$:NEXTN:TIME
R=0
```

```
25 REM ACTUALIZACION
30 B=RND(3)-2:A=A+B:K=INT(TIMER/
50):J=J+1:IF K=>1200 THEN 400 EL
SE IF J=>1000 THEN 500
35 REM COMPROBACION DEL LIMITE D
E POSICION
40 IF A>20 THEN A=20
50 IF A<1 THEN A=1
55 REM PRINT CARRETERA
60 PRINT TAB(A);A$;
65 REM COMPROBACION DE MOVIMIENT
O
70 B$=INKEY$:IF B$="" THEN 70 EL
SE F=ASC(B$):E=E+( (F=8)-(F=9))
75 REM CAMBIO DE MARCHAS
80 IF F=32 AND M=0 THEN POKE &HF
FD7,0:M=1:GOTO 100
90 IF F=32 AND M=1 THEN POKE &HF
FD6,0:M=0
95 REM DETECCION DE COLISIONES
100 IF PEEK(1024+224+E)<>128 THE
N GOSUB 200
105 REM MOVIMIENTO
110 D=RND(11)-1:C=D+A:PRINT @ (4
80)+C,CHR$(159);:PRINT @ (192+L)
,CHR$(128);:PRINT @ (224+E),"u";
:PRINT @ 52,"T       D";:PRINT @
53, USING "####";K;:PRINT @59,US
ING "####";J;:PRINT @ 511, "";:P
LAY"T255;O4;B"
120 GOTO 30
195 REM COMPROBAR LA SIGUIENTE P
OSICION
200 H=PEEK(1024+224+E)
205 REM AMARILLO
210 IF H=159 THEN I=I+1:SOUND 50
,1
215 REM SALIDA DE PISTA
220 IF H=96 THEN I=I+5:SOUND 100
,1
225 REM COMPROBACION DE DA/OS
230 IF I>=50 THEN 300
```

```
240 RETURN
295 REM EXCESO DE DA/OS
300 CLS:PRINT"TU COCHE SE HA HEC
HO PEDAZOS. HAS RECORRIDO";K;"ME
TROS EN";J;"SEGUNDOS":GOTO 1000
395 REM FIN DE TIEMPO
400 CLS:PRINT"FIN DE TIEMPO. HAS
 RECORRIDO";J;"METROS":GOTO 1000
495 REM USTED GANA
500 CLS:PRINT"HAS RECORRIDO LOS
MIL METROS EN";J;"SEGUNDOS":GOTO
 1000
995 REM REINICIALIZAR VELOCIDAD
1000 POKE &HFFD6,0
```

Capítulo VI

LA REALIZACION DE LABERINTOS

Los juegos en los que se deben recorrer laberintos forman una sólida parte de la tradición informática. Ofrecen, infinitas posibilidades para la imaginación del programador y largas horas de frustración para el jugador. Siempre está uno convencido de que si sigue jugando cinco minutos más, resolverá todas sus dificultades. Los escenarios en los que se desarrollan este tipo de juegos son muchos y muy variados. Están tomados tanto de la vida real como de la fantasía. Sin embargo, los principios fundamentales en los que se basan son muy sencillos. El campo de juego es una matriz que puede tener dos dimensiones en los juegos sencillos o tres o más dimensiones en los juegos más complejos.

UN JUEGO QUE UTILIZA UNA MATRIZ BIDIMENSIONAL

Empezaremos por dibujar una simple cuadrícula de 5 × 5 elementos. Es decir, mediante una instrucción DIM definiremos una matriz numérica de dos dimensiones que contendrá 25 elementos.

```
5 REM ESCOGER LAS DIMENSIONES
10 CLS:X=5:Y=5
15 REM DIMENSIONAR LA MATRIZ
20 DIM A(Y,X)
```

Podemos hacer referencia a cualquier punto de la cuadrícula (Figura 6.1) especificando el elemento correspondiente de la matriz mediante los subíndices correspondientes de la misma forma que se lee un mapa cuadriculado. Por ejemplo, la esquina superior izquierda es A(1,1) y la

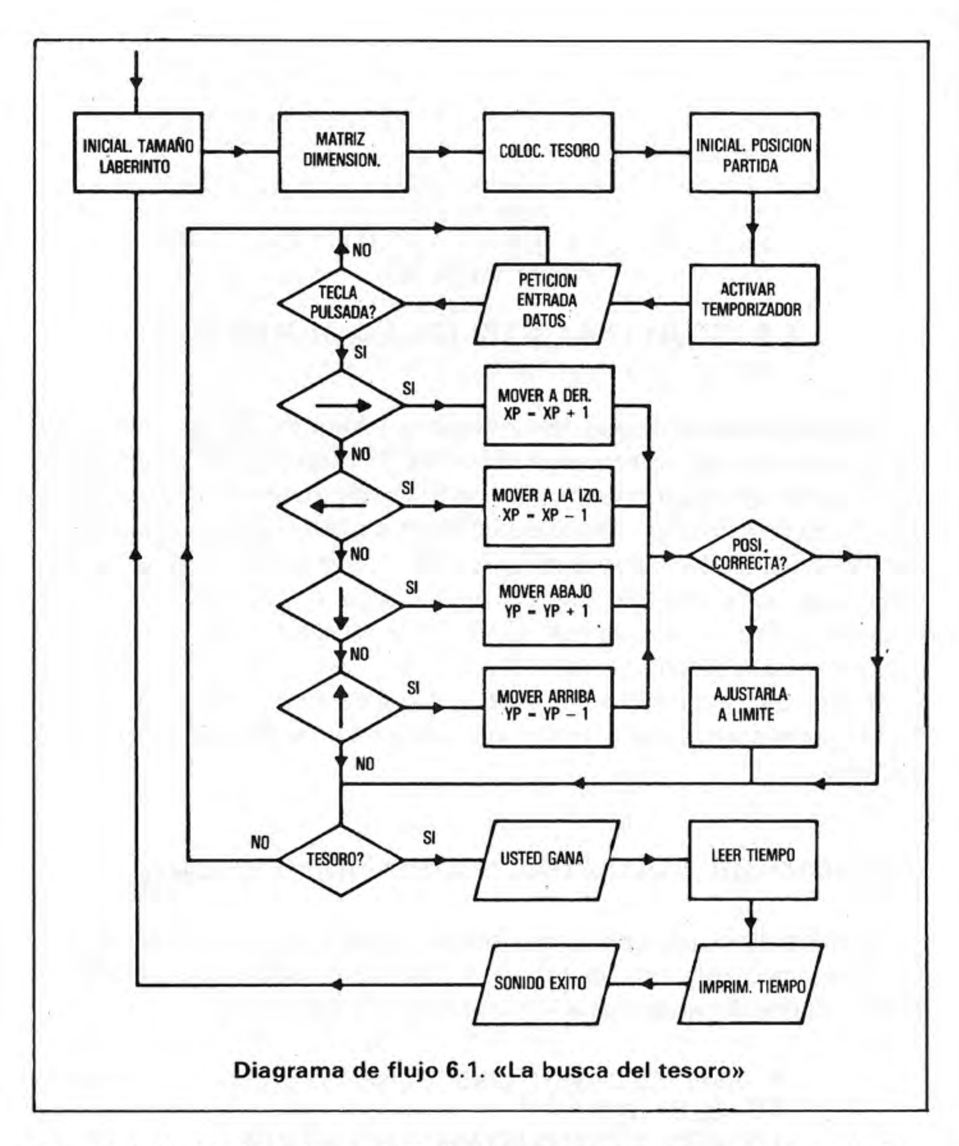


Diagrama de flujo 6.1. «La busca del tesoro»

esquina inferior derecha es A(5,5). Podemos emplear, por tanto, la fórmula general A(Y,X) para definir cualquier punto. Si partimos del centro de la cuadrícula, A(3,3), nos podemos mover hacia el Norte, el Sur, el Este o el Oeste con tan sólo añadir o restar valores a Y o a X. Obsérvese que la Y va antes que la X en la instrucción DIM puesto que el formato es (filas, columnas).

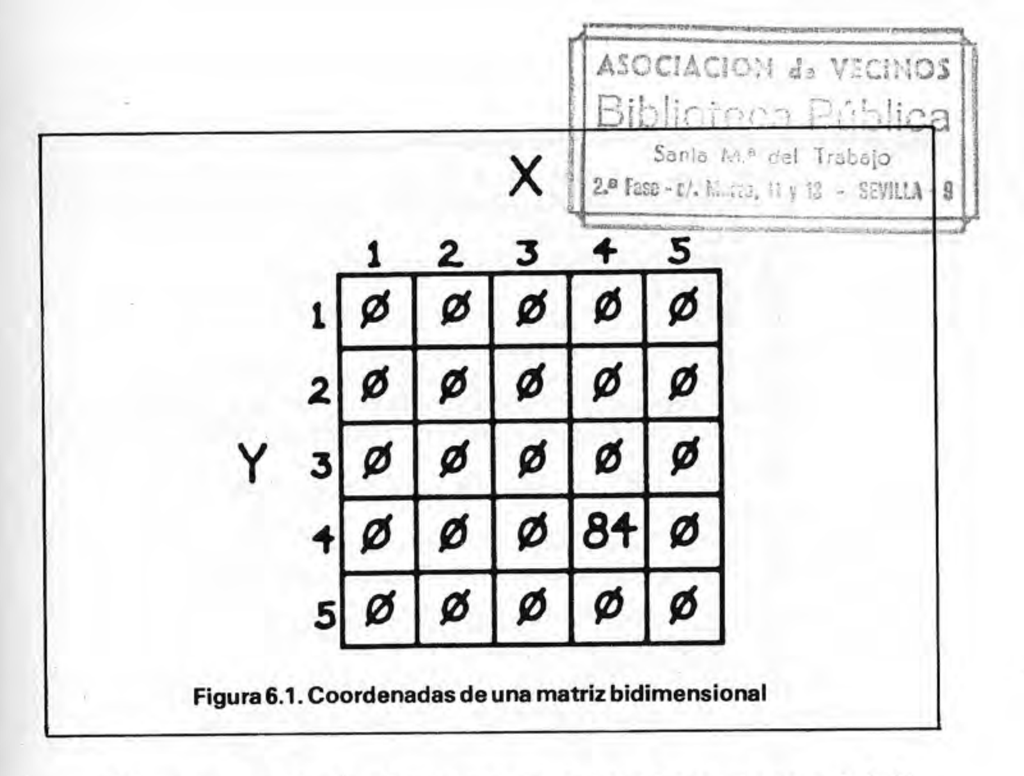


Figura 6.1. Coordenadas de una matriz bidimensional

— Para moverse hacia el Norte restamos 1 de Y para alcanzar A(2,3).
— Para moverse hacia el Sur añadimos 1 a Y para alcanzar A(4,3).
— Para moverse al Oeste restamos 1 a X para alcanzar A(3,2).
— Para moverse hacia el Este añadimos 1 a X para alcanzar A(3,4).

También se pueden realizar movimientos en diagonal combinando dos movimientos sencillos (para ir hacia el Noroeste restamos 1 a Y y 1 a X) y así alcanzamos A(2,2).

Si añade las siguientes líneas conseguirá el listado completo de un sencillo juego de búsqueda del tesoro que utiliza una matriz bidiménsional (Diagrama de flujo 6.1). Por medio de las teclas del cursor, el jugador se puede mover a ciegas por el laberinto buscando con toda la rapidez de que sea capaz, el tesoro invisible. El tesoro está guardado en una posición aleatoria T (código 84). Las variables XP e YP guardan la posición del jugador de tal manera que, como él no existe en el vector, no resulta necesario contar con una rutina de borrado de sus movimientos. De todas formas, sí que se realiza la comprobación de que el jugador no ha rebasado los límites del vector.

Listado completo de «La búsqueda del tesoro»

```
5 REM FIJAR EL TAMA/O
10 CLS:X=5:Y=5
15 REM DIMENSIONAR MATRIZ
20 DIM A(Y,X)
25 REM COLOCAR EL TESORO
30 A(RND(X),RND(Y))=84
35 REM INICIALIZAR COORDENADAS
40 XP=RND(X):YP=RND(Y)
45 REM INICIALIZAR TEMPORIZADOR
50 TIMER=0
55 REM PETICION DE ENTRADA DE IN
FORMACION
60 PRINT @ 200,"DIRECCION?"
65 REM ACTUALIZACION DE LA POSIC
ION
70 A$=INKEY$:IF A$="" THEN 70 EL
SE A=ASC(A$)
80 IF A=9 OR A=93 THEN XP=XP+1 E
LSE IF A=8 OR A=21 THEN XP=XP-1
90 IF A=94 OR A=95 THEN YP=YP-1
ELSE IF A=10 OR A=91 THEN YP=YP+
1
95 REM COORDENADAS CORRECTAS?
100 IF XP<1 THEN XP=1 ELSE IF XP
>X THEN XP=X
110 IF YP<1 THEN YP=1 ELSE IF YP
>Y THEN YP=Y
115 REM ERROR
120 IF A(XP,YP)<>84 THEN PRINT @
 270,"FALLASTE!":SOUND 1,5:CLS:G
OTO 60
125 REM ACIERTO
130 CLS:PRINT @ 225,"FELICIDADES
","HAS ENCONTRADO EL TESORO EN",
(TIMER/50);"SEGUNDOS":SOUND 155,
50:RUN
```

Si así esto le resulta demasiado difícil puede añadir a la línea que le pregunta la dirección en que quiere moverse, una instrucción que imprima las coordenadas de su posición en un momento dado.

```
60 PRINT @ 200,"DIRECCION?";"X";
XP;"Y";YP
```

UN PROGRAMA GENERAL: «LA MAQUINA DE HACER LABERINTOS» (EN CUATRO DIMENSIONES)

Ya que con el mismo campo y principios de juego podemos desarrollar juegos distintos ¿por qué no preparar un programa general (la «máquina de hacer laberintos») que pueda ser fácilmente modificado para ajustarse a las nuevas normas de juego? Además de matrices bidimensionales también podemos utilizar matrices tridimensionales (piénsese en el cubo de la figura 6.2). En este caso, empleamos tres subíndices X, Y y Z. Ahora también podemos movernos hacia arriba y hacia abajo cambiando la Z.

```
10 CLS:X=10:Y=10:Z=5
20 DIM A(Y,X,Z)
```

No importa en qué posición se coloca la Z ya que en este contexto las nociones de arriba y abajo son ficticias.

LA UTILIZACIÓN DE MATRICES TRIDIMENSIONALES

Aunque el manual del Dragón sólo menciona matrices de dos dimensiones, se pueden utilizar otras tridimensionales o de más dimensiones mientras no haya problemas de memoria. Esta matriz tridimensional tiene ahora 500 elementos y ocupa 3648 bytes. Resulta un tanto difícil representar más de tres dimensiones (imagínese tres cubos ocupando el mismo espacio al mismo tiempo) pero si se fantasea un poco acerca de los viajes en el tiempo, siempre se podrá definir una matriz que tenga los subíndices X, Y, Z y T.

```
10 CLS:X=10:Y=10:Z=5:T=2
20 DIM A(Y,X,Z,T)
```

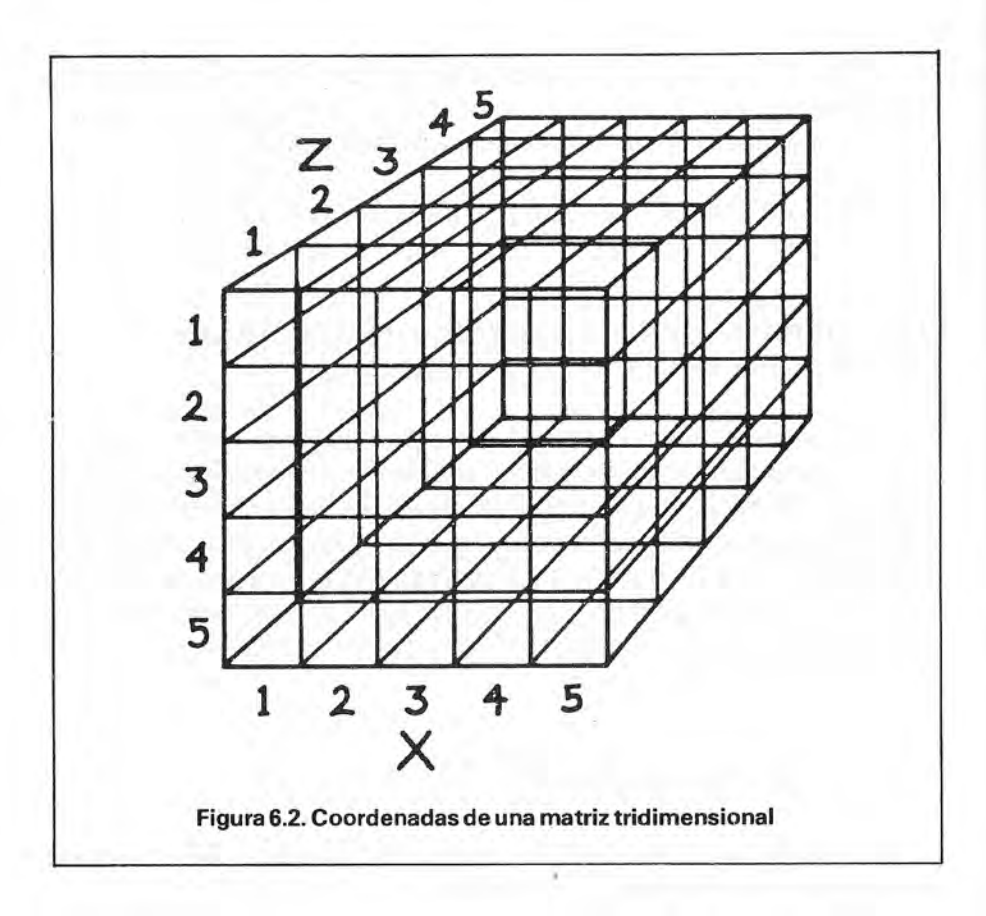


Figura 6.2. Coordenadas de una matriz tridimensional

Esto realmente se come la memoria (10910 bytes) y ocupa, con seguridad, bastante más de lo que se hubiera esperado. El secreto está en que la cuarta dimensión tiene tres de profundidad incluso a pesar de que hemos dado a T el valor 2 y el cero está automáticamente permitido siempre en cualquier matriz que se declare. Si se anda escaso de memoria siempre se puede restar 1 a cada dimensión. Normalmente no nos preocuparemos mucho de estas cuestiones cuando trabajemos con matrices pequeñas ya que resulta más difícil recordar hacia dónde nos estamos dirigiendo. En este programa únicamente hemos ofrecido la opción de poner a cero una sola dimensión: la T.

Cuando se crea una matriz está completamente llena de ceros. Tenemos que cambiar esta situación para conseguir un programa de juego. La matriz acostumbra a llenarse, de forma aleatoria, con un nú-

mero aleatorio de objetos aleatorios (que se representan dentro de la matriz por medio de números). Hay muchas maneras distintas de hacerlo. En algunas, se vuelve a escribir en posiciones ya ocupadas. También es posible tratar cada punto separadamente recurriendo a bucles FOR NEXT. Para tomar la decisión de colocar o no un número, se puede recurrir a RND (2)—1 (que da 0 o un 1) y, según el número que resulte, colocar o no un número. Este último número puede conseguirse haciendo RND (100).

```
10 CLS:X=10:Y=10:Z=5:T=2
20 DIM A(Y,X,Z,T)
30 FOR N1=1 TO Y:FOR N2=1 TO X:F
OR N3=1 TO Z:FOR N4=0 TO T
40 A(N1,N2,N3,N4)=(RND(2)-1)*(RN
D(100))
50 NEXT N4,N3,N2,N1
```

LA FORMACIÓN DE UNA MATRIZ, VISTA EN LA PANTALLA

Si se quiere ver cómo se va llenando gradualmente la matriz, empléese la siguiente línea 42 (que puede añadir provisionalmente al programa). Con ella se tendrá en pantalla una presentación en movimiento de la formación de la matriz.

```
42 PRINT @ 0,"x";X;N1;"y";Y;N2;"
z";Z;N3;"t";T;N4
```

La tarea inmediata consiste en escoger una posición aleatoria desde la que empezar a jugar. Hemos de determinar las cuatro coordenadas.

```
60 XP=RND(X):YP=RND(Y):ZP=RND(Z)
:TP=RND(T)
```

Su posición inicial es ahora A(YP, XP, ZP, T).

La selección aleatoria de objetos numerados del 1 al 100 que se obtiene, normalmente no contendrá a todos esos números. Es muy probable también que se desee colocar algunos objetos esenciales —como la salida— en lugares determinados. Se puede tratar este tipo de objetos se-

paradamente de forma similar a cómo se colocaron los otros objetos. En esta ocasión, los números utilizados serán mayores de 100 para evitar confusiones. La *x* minúscula tiene el código ASCII 120; por tanto, utilicémosla para indicar la salida.

```
70 A(RND(X),RND(Y),RND(Z),RND(T)
)=120
```

Si se necesitara más de una salida para el laberinto bastaría con que añadiera un bucle FOR-NEXT de longitud XT que englobara el procedimiento de selección. Habría que añadir también la variable XT a las definidas en la línea 10.

```
10 CLS:X=5:Y=5:Z=2:T=2:XT=2
70 FOR N=1 TO XT:A(RND(X),RND(Y)
,RND(Z),RND(T))=120:NEXT N
```

Para controlar los movimientos podemos utilizar como antes las teclas y una comprobación lógica. Esta vez emplearemos las letras N, S, E y W (Norte, Sur, Este y Oeste) y U, D y F (Arriba, Abajo y Adelante respectivamente) en vez de las teclas de control del cursor. La disposición de las comprobaciones dentro de la línea es significativa. Lo primero que se hace es comprobar que no haya ninguna tecla pulsada. Así se logra que la reacción sea lo más rápida posible. Las otras comprobaciones se han dispuesto en orden de mayor a menor uso.

```
80 M$=INKEY$:IF M$="" THEN 80 EL
SE XP=XP+(M$="E")-(M$="W"):YP=YP
+(M$="S")-(M$="N"):ZP=ZP+(M$="U"
)-(M$="D"):TP=TP+(M$="F")-(M$="B
")
```

LAS COMPROBACIONES EN LAS RUTINAS DE MOVIMIENTO PARA NO REBASAR LOS LIMITES

Si el jugador realiza un movimiento que lo lleva fuera de los límites declarados de la matriz, el programa acabará anormalmente con un error BS (Subíndice incorrecto) al haberse intentado evaluar el conteni-

do de un elemento de la matriz situado más allá de los límites verdaderos. Hay que escribir siempre dentro de las rutinas de movimiento para que no se traspasen los límites fijados. En la búsqueda del tesoro realizábamos estas comprobaciones mediante IF-THEN. Sin embargo, resulta más rápido multiplicar el cambio de posición deseado por una comprobación lógica realizada sobre los límites inferior y superior de la matriz. Por ejemplo, XP = XP + (M$ = «E»)* (XP < X) significa que, si se pulsa la tecla E, XP aumentará de valor sólo en el caso de que XP sea menor que el valor máximo. Podemos añadir un salto hacia atrás al final de la línea al que sólo se llegará si se ha pulsado una tecla incorrecta.

```
80 M$=INKEY$:IF M$="" THEN 80 EL
SE XP=XP+(M$="E")*(XP<X)-(M$="W"
)*(XP>0):YP=YP+(M$="S")*(YP<Y)-(
M$="N")*(YP>0):ZP=ZP+(M$="U")*(Z
P<Z)-(M$="D")*(ZP>0):TP=TP+(M$="
F")*(TP<T)-(M$="B")*(TP>0):ELSE
80
```

Para probar el funcionamiento del programa en este momento, se puede añadir otra línea provisional que imprima el contenido de la matriz en la posición que se ocupe entonces. También debe imprimirse el valor de las coordenadas de la posición (mientras que el jugador prueba, una a una, las diversas teclas que controlan los movimientos). Si durante el desarrollo de una partida se quiere saber cuál es su posición, se puede emplear una línea similar.

```
82 PRINT @ 0,"NO";A(YP,XP,ZP,TP)
;"X";XP;"Y";YP;"Z";ZP;"T";TP:GOT
O80
```

LA RUTINA DE DECODIFICACION

Ahora que ya hemos creado nuestro sistema de preparación de laberintos, podemos pensar en cómo convertir los números que tenemos colocados dentro de la matriz en objetos utilizados durante el juego. Esto requerirá una rutina de decodificación. Aunque es posible contar con un total de 100 objetos diferentes, se acostumbra a jugar con

un número más reducido. Por tanto, dividiremos los 100 números en varios bloques de tamaño variable (S). S está definida en la línea 10 y puede cambiarse para conseguir un número distinto de objetos. En la misma línea se coloca un salto múltiple del tipo ON GOTO. Sólo se alcanzan los puntos de llegada correspondientes a los veinte objetos distintos representados en esta línea si S = 5 ya que un valor superior de S haría que sólo se alcanzaran los primeros puntos de llegada de los saltos. Si (A(YP, XP, ZP, T)/S) es menor que 1 o si S es menor que 5, el control pasará a la rutina de actualización (la línea 100) sin que se haya detectado ningún objeto.

```
10 CLS:X=5:Y=5:Z=2:T=2:XT=2:S=5
90 ON (A(YP,XP,ZP,TP)/S) GOTO 11
00,1200,1300,1400,1500,1600,1700
,1800,1900,2000,2100,2200,2300,2
400,2500,2600,2700,2800,2900,300
0
```

LA RUTINA DE ACTUALIZACION

Después de cada movimiento es necesario saltar a una rutina de actualización que incremente el tiempo (TI) y la distancia (DI) al mismo tiempo que decrementa la comida consumida (FO). El temporizador TIMER no puede recibir el valor 0 hasta que el laberinto no esté construido. Por tanto, colocaremos la instrucción de inicialización del temporizador al final de la imprescindible rutina de colocación de objetos en el laberinto.

```
70 FOR N=1 TO XT:A(RND(X),RND(Y)
,RND(Z),RND(T))=120:NEXT N:TIMER=
0
100 TI=TI+(TIMER/50):DI=DI+1:FO=
FO-1
```

Como presumiblemente, se querrá imponer algún límite al tiempo o a la cantidad de comida, tendremos que incluir las instrucciones de comprobación adecuadas que hagan saltar la ejecución a los puntos del programa que se encarguen de las consecuencias de rebasar los límites.

```
100 TI=TI+(TIMER/50):DI=DI+1:FO=
FO-1:IF TI>6000 THEN 4000 ELSE I
F FO <1 THEN 5000
```

Lo que pase en las distintas rutinas asociadas a cada tipo de objeto es la parte realmente interesante de la programación y queda enteramente a su elección. Sin embargo, todas las rutinas deben volver a la rutina de actualización. Volveremos más adelante sobre este punto.

UNA VENTANA EN EL LABERINTO

Aunque acabamos de construir un complejo laberinto en cuatro dimensiones para la «máquina de hacer laberintos», no podemos ver hacia dónde nos estamos dirigiendo: sólo podemos reaccionar ante las distintas situaciones cuando éstas ya se han dado. En un laberinto verdadero deberíamos gozar de una visión limitada de los alrededores. A partir de ella podríamos decidir en qué dirección movernos. El programa siguiente se diferencia bastante del anterior especialmente en el hecho de que utiliza las 15 líneas superiores de la pantalla para presentar el campo de juego pero ofrece un ventana móvil que sólo deja ver una pequeña parte del laberinto. Los distintos objetos se representan por bloques gráficos de diferentes colores.

Una pantalla de texto del Dragón contiene 512 posiciones de impresión. Las representaremos mediante una sola matriz unidimensional. A esta matriz le daremos una dimensión de 672 elementos para dejar tres líneas adicionales en la parte superior e inferior de la pantalla. Lo hacemos así por que deseamos colocar la ventana (que es de 6 × 6 unidades) de forma que su centro pueda estar en cualquier punto de la pantalla. Si nuestra matriz no fuera lo suficientemente grande, acabaríamos provocando errores BS. Para hacer que el programa sea más fácilmente modificable, vamos a definir variables que permitan especificar los parámetros de la matriz.

A1 = 672 (tamaño de la matriz).
A2 = 97 (inicio de la pantalla).
A3 = 576 (final de la pantalla).
A4 = 447 (parte de la pantalla que utilizaremos).

A5 = 450 (bloques).
A6 = 50 (objetos 1).
A7 = 50 (objetos 2).

Como la rutina de inicialización es bastante compleja y sólo se emplea una vez en cada partida, la colocamos al final del programa. Se accede a ella saltando desde la línea 10.

```
10 CLS:GOTO 12000
12000 A1=672:A2=97:A3=576:A4=447
:A5=450:A6=50:A7=50:DIM B(A1)
```

Primero rellenamos de negro toda la matriz con CHR$,(128) ya que de otra forma, recibiríamos la sorpresa de descubrir que CHR$(0) no existe.

```
12010 FOR A=1 TO A1:B(A)=128:NEX
T A:SOUND 1,1
```

Obsérvese que hemos añadido un poco de sonido al final de cada parte de esta rutina. De esta forma tendremos la seguridad de que realmente está ocurriendo algo mientras esperamos que se coloquen en la matriz los distintos elementos.

El programa intenta luego colocar aleatoriamente un bloque verde (143) durante A5 (450) veces. Obsérvese lo útil que resulta aquí RND(0). RND(0) dará como resultado un número entre el 0 y el 1 que, multiplicado por A4 (el número de posiciones que tiene la pantalla), nos dará números comprendidos entre el 0 y el A4. Añadimos A2 (el inicio de la pantalla) a esos números y el resultado que obtenemos lo utilizamos para definir los elementos de la matriz B (no es necesario redondear estos números ya que automáticamente se emplea el elemento más cercano).

```
12020 FOR A=1 TO A5:B(RND(0)*A4+
A2)=143:NEXT A:SOUND 10,1
```

Para elaborar un poco más las cosas también podemos colocar bloques de color aleatorio mediante (RND(7)*16) + 143 ya que los colores tienen códigos mayores que el 143 que van de 16 en 16.

```
12030 FOR A=1 TO A6:B(RND(0)*A4+
A2)=(RND(7)*16)+143:NEXT A:SOUND
 20,1
```

La línea siguiente es un poco malvada pues coloca trampas invisibles al guardar el número 144 en distintas posiciones de la matriz. Tengamos en cuenta que CHR$(144) es otro bloque negro más: no se puede distinguir de CHR$(128), el que habíamos utilizado para construir las paredes del laberinto.

```
12040 FOR A=1 TO A7:B(RND(0)*A4+
A2)=144:NEXT A:SOUND 30,1
```

Como toque final haremos que la parte de la matriz correspondiente a la línea inferior del laberinto (elementos 513 a 544) sea verde para darle al jugador alguna posibilidad de alcanzar la salida (XT) (que se encuentra en un determinado lugar de la pared inferior y se indica por una X).

```
12050 FOR A=513 TO 544:B(A)=143:
NEXT A:SOUND 40,1:XT=545+RND(29)
:B(XT)=88
```

Tanto el jugador (código 85, U) como un dragón (código 100, D en inversa) quedan situados de forma aleatoria dentro del laberinto.

```
12060 YU=85:U=RND(A4)+A2:B(U)=YU
:D=RND(A4)+A2:B(D)=100:SOUND 50,
1
```

Para dar la impresión de que somos justos, incluiremos una comprobación que asegure que, después de examinar los cuatro cuadrados que rodeen su posición, es posible, al menos, un primer movimiento hacia un cuadrado verde. Los cuadrados de la izquierda y de la derecha corresponden a +1 y −1 pero los de arriba y de abajo corresponden a +32 y −32 puesto que hay que desplazarse una línea completa hacia arriba o hacia abajo. Se hace una segunda comprobación para ver que el dragón no está totalmente rodeado de paredes.

Si cualquiera de estas dos comprobaciones da resultado negativo, se

pone un cuadrado verde en las posiciones escogidas anteriormente y se pasa a escoger otras nuevas.

```
12070 IF (B(U+1)=143 OR B(U-1)=1
43 OR B(U+32)=143 OR B(U-32)=143
) AND (B(D+1)=143 OR B(D-1)=143
OR B(D+32)-143 OR B(D-32)=143) T
HEN 12080 ELSE SOUND 1,1:B(U)=14
3:B(D)=143:GOTO 12060
```

Ahora que hemos llenado totalmente la matriz a nuestro gusto, podemos examinarla por completo imprimiendo cada elemento como su representación en forma de carácter mediante CHR$(B(A)). Tras un bucle de retardo, se asigna un nuevo valor al temporizador TIMER y se dan los valores iniciales a las variables correspondientes al límite de tiempo (TS = 600 (10 minutos)), cantidad de comida (FO) y dinero (MO) antes de volver a la rutina principal.

```
12080 CLS0:FOR A=A2 TO A3:PRINT
CHR$(B(A));:NEXT A:FOR A=1 TO 10
00:NEXT A:TIMER=0:TS=600:FO=100:
MO=100:GOTO 100
```

La primera tarea que debe realizar la rutina principal es borrar la pantalla. Luego hay que calcular los límites de la ventana e imprimir la parte correspondiente de la matriz. Las variables X e Y definen las posiciones situadas tres líneas por encima y por debajo de la posición del jugador. El bucle FOR-NEXT interior imprime seis caracteres desplazándose cada vez una posición de impresión y un elemento de la matriz. El bucle exterior mueve la posición de impresión y el elemento de la matriz de línea a línea.

```
100 CLS
```

```
110 X=U-99:Y=U+99:Z=Y-X:FOR A=0
TO Z STEP 32:FOR C=0 TO 6:PRINT
@ (A+C),CHR$(B(X+A+C)):NEXT C,A
```

Querremos saber qué tal nos va, así que añadamos debajo de la ventana información acerca del juego.

```
120 PRINT @ 224,"TIEMPO",,"MOVIM
IENTOS";M,,"COMIDA";FO;,,"DINERO
";MO
```

Ahora viene la comprobación de teclas y de que no hemos rebasado el límite de tiempo.

```
130 I$=INKEY$:IF I$="" THEN PRIN
T @ 230,TIMER/50:IF TIMER/50>TS
THEN 1000 ELSE 130
```

La dirección de movimiento es relativa a la posición (U) ocupada en un momento dado por el jugador y se calcula mediante una comprobación lógica (*32 donde necesite moverse de línea a línea). La posición que se quiere alcanzar es V. La variable de control de movimientos M se incrementa al mismo tiempo que la variable que controla la cantidad de comida FO se decrementa.

```
140 I=ASC(I$):V=U+32*(I=94 OR I=
95)-32*(I=10 OR I=91)+1*(I=8 OR
I=21)-1*(I=9 OR I=93):M=M+1:FO=F
O-1
```

Ahora se comprueba si todavía queda comida. Luego se ve si no hay nada en nuestro camino (a menos que sea un bloque verde, claro está). Si todo está despejado, el elemento de la matriz que representa la antigua posición del jugador se llena con el número 143, la posición del jugador en la matriz se cambia por la nueva y finalmente se comprueban las teclas otra vez.

```
150 IF FO<1 THEN 2000
160 IF B(V)<>143 THEN 170 ELSE B
(U)=143:U=V:B(U)=YU:GOTO 110
```

Si el camino no está libre, hemos de detectar qué es lo que hay en la siguiente posición: la salida (88), las paredes (128), las paredes falsas

(144) o el dragón (100). Una vez sabemos de qué se trata, hay que saltar a las subrutinas correspondientes. Recuerde que cuando cualquier subrutina llega a su final debe volver de nuevo a la línea 110.

```
170 IF B(V)=88 THEN 3000
180 IF B(V)=128 THEN PRINT @ 384
,"CERRADO EL PASO":SOUND 1,5:GOT
O 110
190 IF B(V)=144 THEN 3000
200 IF B(V)=100 THEN 4000
```

FORMA DE EVITAR CARACTERES NO PERMITIDOS EN UNA MATRIZ

Se incluye una «red de seguridad» —como para los trapecistas— por si se da el caso de haber introducido en la matriz un carácter no permitido (antes de haber preparado los saltos a las distintas subrutinas para cada color en la línea del ON-GOTO). Esta «red» coge el contenido de la posición especificada de la matriz, le resta 143 y luego la divide por 16 para obtener números comprendidos entre el 1 y el 7. Al principio, resulta sorprendente ver cómo se puede jugar con estos números para que hagan lo que uno quiere. Sin embargo, comprobará que un poco de reflexión ahorra mucho tiempo y espacio. Sin la «red de seguridad» un carácter alfanumérico —para el que no hemos previsto ninguna comprobación en las líneas anteriores— puede causar un error FC puesto que la expresión evaluada por el ON-GOTO sería negativa pues los códigos de todas las teclas son menores que 143 y, por tanto, incorrectos.

```
210 IF B(V)<144 THEN 110
220 ON (B(V)-143)/16 GOTO 5000,6
000,7000,8000,9000,10000,11000
```

Consideraremos más adelante las consecuencias de las diferentes subrutinas posibles.

AHORA ES SU TURNO

1) Vuelva a escribir la «búsqueda del tesoro» con el mínimo número de líneas posible. Emplee las técnicas explicadas en programas posteriores a ése.
2) Cambie la «máquina de hacer laberintos» de forma que tenga que pasar por una rutina de comprobación si quiere viajar por la cuarta dimensión.
3) Escriba una rutina de movimientos alternativa para «la máquina de hacer laberintos» que emplee mandos para juego en vez de teclas (no es tan sencillo como parece).
4) Cambie el tamaño de la ventana del laberinto a 8 × 8 en vez de 6 × 6.
5) Diseñe una rutina que se encargue de gestionar el funcionamiento de la ventana en la «máquina de hacer laberintos».

Listado completo de la «máquina de hacer laberintos»

```
5 REM TAMA/O DE LA MATRIZ
10 CLS:X=5:Y=5:Z=2:T=2:XT=2:S=5
15 REM INICIALIZAR LA MATRIZ
20 DIM A(Y,X,Z,T)
25 REM LLENAR LA MATRIZ
30 FOR N1=1 TO Y:FOR N2=1 TO X:F
OR N3=1 TO Z:FOR N4=0 TO T
40 A(N1,N2,N3,N4)=(RND(2)-1)*(RN
D(100))
41 REM COMPROBACION DE LA FORMAC
ION DE LA MATRIZ
42 PRINT @ 0,"x";X;N1;"y";Y;N2;"
z";Z;N3;"t";T;N4
50 NEXT N4,N3,N2,N1
55 REM POSICION INICIAL
60 XP=RND(X):YP=RND(Y):ZP=RND(Z)
:TP=RND(T)
65 REM OBJETOS ESENCIALES
70 FOR N=1 TO XT:A(RND(X),RND(Y)
,RND(Z),RND(T))=120:NEXT N:TIMER
=0
```

```
75 REM MOVIMIENTO
80 M$=INKEY$:IF M$="" THEN 90 EL
SE XP=XP+(M$="E")*(XP<X)-(M$="W"
)*(XP>0):YP=YP+(M$="S")*(YP<Y)-(
M$="N")*(YP>0):ZP=ZP+(M$="U")*(Z
P<Z)-(M$="D")*(ZP>0):TP=TP+(M$="
F")*(TP<T)-(M$="B")*(TP>0):ELSE
80
81 REM CONTENIDO DE LA MATRIZ Y
COMPROBACION DE MOVIMIENTOS
82 PRINT @ 0,"NO";A(XP,YP,ZP,TP)
;"X";XP;"Y";YP;"Z";ZP;"T";TP:GOT
O80
85 REM DECODIFICACION DE OBJETOS
90 ON (A(XP,YP,ZP,TP)/S) GOTO 10
00,1100,1200,1300,1400,1500,1600
,1700,1800,1900,2000,2100,2200,2
300,2400,2500,2600,2700,2800,290
0
95 REM SI NO ES NINGUN OBJETO IR
 A ACTUALIZACION
100 TI=TI+(TIMER/50):DI=DI+1:FO=
FO-1:IF TI>6000 THEN 4000 ELSE I
F FO <1 THEN 5000
995 REM RUTINAS PARA LOS OBJETOS
1000 .....GOTO 100

     <> etc

2900 .....GOTO 100
3995 REM FIN DE TIEMPO
4000 .....END
4995 REM FIN DE COMIDA
5000 .....END
```

Listado completo de la ventana en el laberinto

```
5 REM SALTAR A LA RUTINA DE INIC
IALIZACION
10 CLS:GOTO 12000
```

```
95 REM RUTINA PRINCIPAL
100 CLS
105 REM IMPRESION DE LA VENTANA
110 X=U-99:Y=U+99:Z=Y-X:FOR A=0
TO Z STEP 32:FOR C=0 TO 6:PRINT
@ (A+C),CHR$(B(X+A+C)):NEXT C,A
115 REM INFORMACION
120 PRINT @ 224,"TIEMPO",,"MOVIM
IENTOS";M,,"COMIDA";FO;,,"DINERO
";MO
125 REM COMPROBACION TECLAS/ACTU
ALIZACION TIEMPO
130 I$=INKEY$:IF I$="" THEN PRIN
T @ 230,TIMER/50:IF TIMER/50>TS
THEN 1000 ELSE 130
135 REM COMPROBACION DE LA DIREC
CION
140 I=ASC(I$):V=U+32*(I=94 OR I=
95)-32*(I=10 OR I=91)+1*(I=8 OR
I=21)-1*(I=9 OR I=93):M=M+1:FO=F
O-1
145 REM COMPROBACION DE LA CANTI
DAD DE COMIDA
150 IF FO<1 THEN 2000
155 REM ACTUALIZACION DE LA POSI
CION SI ESTA LIBRE EL CAMINO
160 IF B(V)<>143 THEN 170 ELSE B
(U)=143:U=V:B(U)=YU:GOTO 110
165 REM COMPROBACION DE LA SALID
A
170 IF B(V)=88 THEN 3000
175 REM COMPROBACION PARED
180 IF B(V)=128 THEN PRINT @ 384
,"CERRADO EL PASO":SOUND 1,5:GOT
O 110
185 REM PARED FALSA
190 IF B(V)=144 THEN 3000
195 REM COMPROBACION DRAGON
200 IF B(V)=100 THEN 4000
205 REM RED DE SEGURIDAD
210 IF B(V)<144 THEN 110
215 REM SALTAR SEGUN LOS COLORES
```

```
220 ON (B(V)-143)/16 GOTO 5000,6
000,7000,8000,9000,10000,11000
995 REM FIN DE TIEMPO
1995 REM FIN DE COMIDA
2995 REM PARED FALSA
3995 REM DRAGON
4995 REM AMARILLO
5995 REM AZUL
6995 REM ROJO
7995 REM NARANJA PALIDO
8995 REM CIAN
9995 REM MAGENTA
10995 REM NARANJA
11990 GOTO 130
11995 REM INICIALIZAR VARIABLES
12000 A1=672:A2=97:A3=576:A4=447
:A5=450:A6=50:A7=50:DIM B(A1)
12005 REM ENNEGRECER LA MATRIZ
12010 FOR A=1 TO A1:B(A)=128:NEX
T A:SOUND 1,1
12015 REM PONER EL VERDE
12020 FOR A=1 TO A5:B(RND(0)*A4+
A2)=143:NEXT A:SOUND 10,1
12025 REM PONER COLORES
12030 FOR A=1 TO A6:B(RND(0)*A4+
A2)=(RND(7)*16)+143:NEXT A:SOUND
 20,1
12035 REM COLOCAR LAS PAREDES FA
LSAS
12040 FOR A=1 TO A7:B(RND(0)*A4+
A2)=144:NEXT A:SOUND 30,1
12045 REM BORRAR INFERIOR Y MARC
AR SALIDA
12050 FOR A=513 TO 544:B(A)=143:
NEXT A:SOUND 40,1:XT=545+RND(29)
:B(XT)=88
12055 REM COLOCAR JUGADOR Y DRAG
ON
12060 YU=85:U=RND(A4)+A2:B(U)=YU
:D=RND(A4)+A2:B(D)=100:SOUND 50,
1
```

```
12065 REM VER SI ES POSIBLE MOVE
RSE
12070 IF (B(U+1)=143 OR B(U-1)=1
43 OR B(U+32)=143 OR B(U-32)=143
) AND (B(D+1)=143 OR B(D-1)=143
OR B(D+32)-143 OR B(D-32)=143) T
HEN 12080 ELSE SOUND 1,1:B(U)=14
3:B(D)=143:GOTO 12060
12075 REM IMPRIMIR EL LABERINTO
Y EMPEZAR
12080 CLS0:FOR A=A2 TO A3:PRINT
CHR$(B(A));:NEXT A:FOR A=1 TO 10
00:NEXT A:TIMER=0:TS=600:FO=100:
MO=100:GOTO 100
```

Capítulo VII

LOS MOVIMIENTOS Y LAS OPCIONES EN LOS JUEGOS

La diferencia más importante que existe entre un juego de aventuras bueno y otro que no lo es, reside en la distinta complejidad de los problemas que en uno y otro se encuentren y en la facilidad con que se puedan resolver. Para desarrollar al máximo nuestra «máquina de hacer laberintos» y convertirla en un juego de aventuras, debemos explicar cómo preparar unas cuantas rutinas que aporten más complejidad y nuevas posibilidades al juego. De esta forma, el lector será capaz de crear su propio juego (de éxito mundial, seguro).

LA FORMA DE DETECTAR LA POSICION DEL JUGADOR

Resulta esencial o, cuando menos, bastante práctico disponer de la posibilidad de detectar qué es lo que hay en las cercanías de la posición del laberinto que ocupamos. Por esta razón vamos a echar una mirada al método que utilizaremos para cubrir esta necesidad (Diagrama de flujo 7.1). Emplearemos como ejemplo el sencillo programa de búsqueda del tesoro.

Ahora sabemos cuál es nuestra posición puesto que se definió en la línea 120 mediante A(YP, XP). Sin embargo, no conocemos cuál es la del tesoro pues, a pesar de que la línea 30 la determinó con A(RND(Y), RND(X)), no almacenamos en ninguna variable las coordenadas obtenidas. Podemos cambiar la línea 30 fácilmente y guardar las coordenadas Y y X en dos nuevas variables TY y TX respectivamente.

```
30 TY=RND(Y):TX=RND(X):A(TY,TX)=
84
```

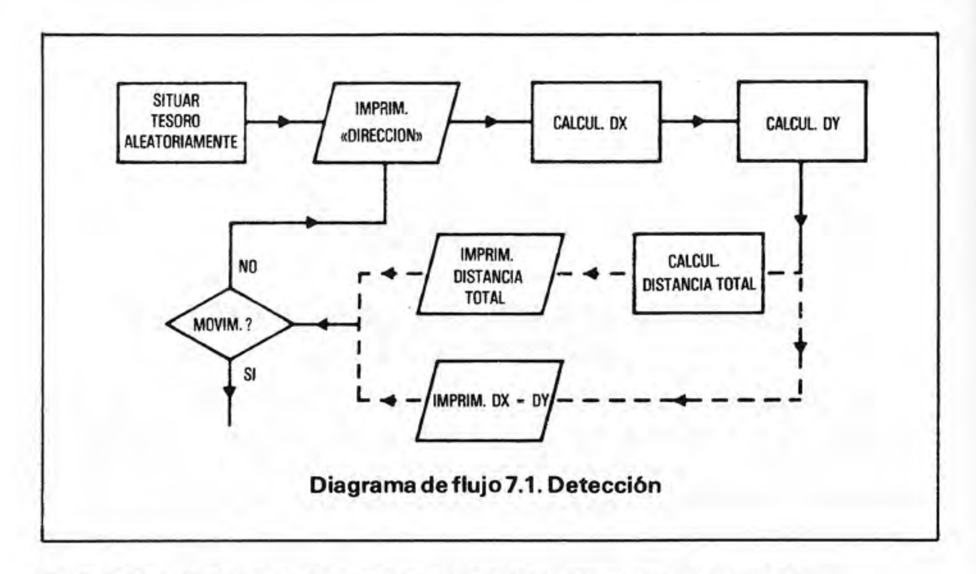


Diagrama de flujo 7.1. Detección

CALCULO DE DISTANCIAS ENTRE DOS POSICIONES, EN UN JUEGO

También resultaría interesante conocer la distancia que nos separa de nuestro objetivo. Por tanto, añadiremos un método para calcular la distancia existente entre nuestra posición en un momento dado y el tesoro. El método tomará la forma de una subrutina a la que se saltará después de preguntar al jugador en qué dirección desea moverse.

```
60 PRINT @ 200,"DIRECCION?":GOSU
B 1000
```

La distancia hasta el tesoro en la dirección del eje X puede calcularse restando TX a XP (Figura 7.1)

```
1000 DX=XP-TX:RETURN
```

Lógicamente, el valor de DX que obtengamos por este método puede ser positivo o negativo según a qué lado del tesoro nos encontremos. Necesitamos recurrir a la función ABS para obtener los valores absolutos. De esta forma DX siempre es positiva.

```
1000 DX=ABS(XP-TX):RETURN
```

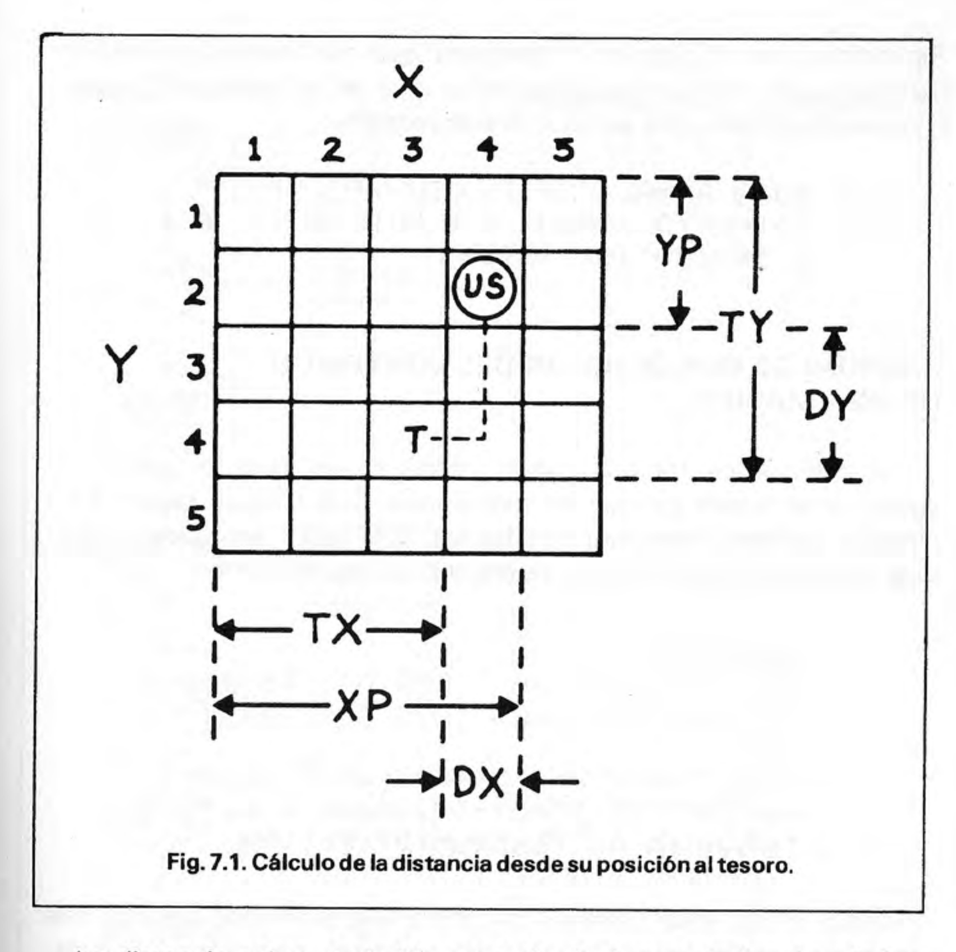


Fig. 7.1. Cálculo de la distancia desde su posición al tesoro.

La distancia sobre el eje Y se calcula de forma similar. Imprimiremos ambas distancias (sobre el eje X y sobre el eje Y) en la esquina superior izquierda de la pantalla.

```
1000 DX=ABS(XP-TX):DY=ABS(YP-TY)
:PRINT @ 0,"DX=";DX;"DY=";DY:RET
URN
```

De esta forma, resulta muy sencillo decidir hacia qué dirección debe uno moverse. Quizás debiéramos calcular y dar una distancia total (DT) al tesoro en vez de dar las dos distancias parciales. La distancia total la calcularemos sumando ambas distancias parciales.

Se indica pues al jugador la distancia que aún tiene que recorrer pero sin darle ninguna pista acerca de cuál es la dirección a tomar. Hay que pensárselo dos veces antes de moverse.

```
1000 DX=ABS(XP-TX):DY=ABS(YP-TY)
:DT=DX+DY:PRINT @ 0,"DISTANCIA A
L TESORO";DT:RETURN
```

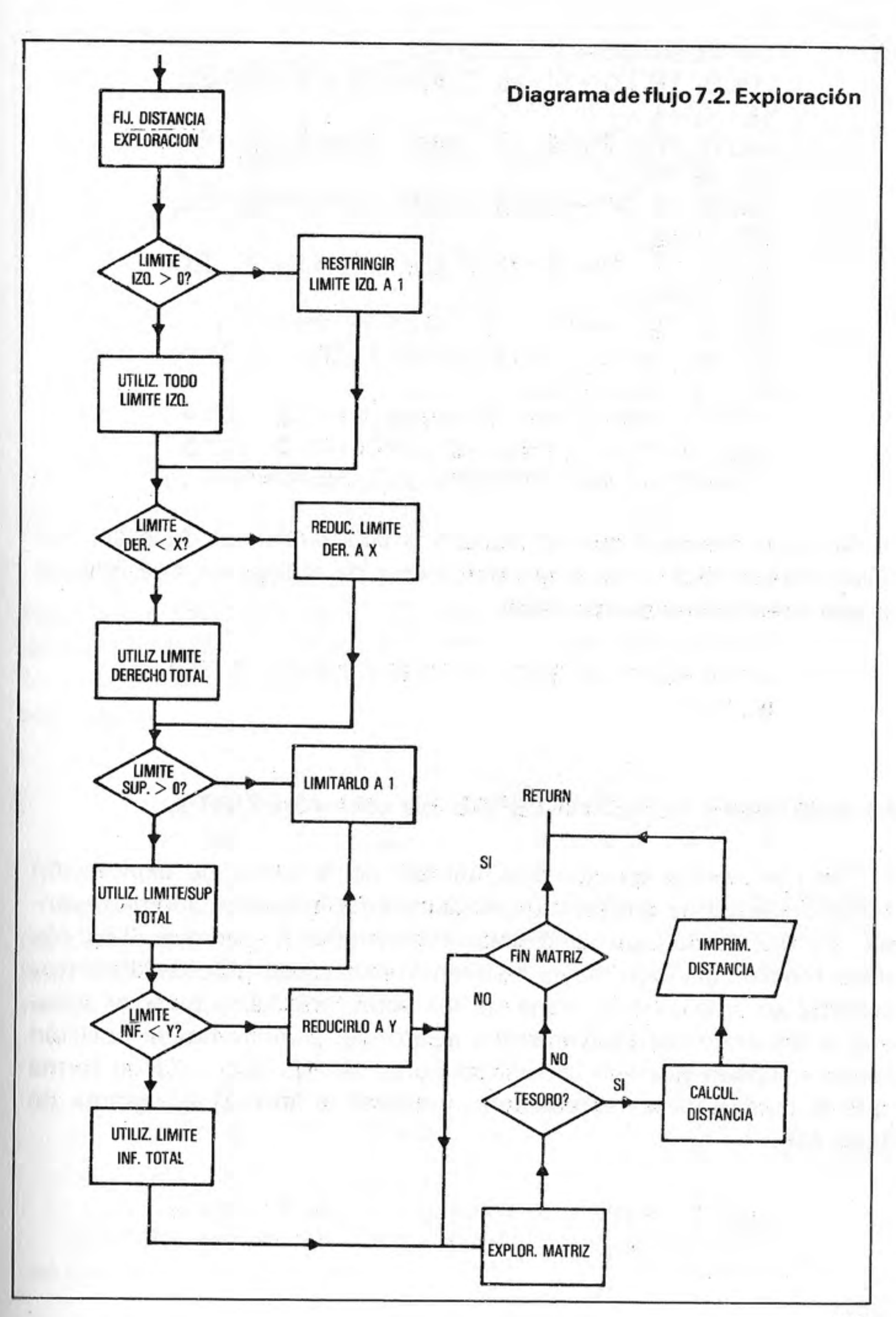
BARRIDO DE EXPLORACION DEL CONTENIDO DE UNA MATRIZ

Si exploramos toda la matriz mediante un barrido podríamos encontrar el tesoro aunque no hubiéramos guardado su posición en ninguna variable. Mediante dos bucles FOR-NEXT anidados podríamos explorar la matriz hasta que encontrarámos el tesoro.

```
1000 REM
1010 FOR N=1 TO Y:FOR M=1 TO X:I
F A(N,M)=84 THEN 1020 ELSE NEXT
M,N
1020 TY=N:TX=M:DX=ABS(XP-TX):DY=
ABS(YP-TY):DT=DX+DY:PRINT @ 0,"D
ISTANCIA AL TESORO";DT:RETURN
```

Desde luego, esta rutina es más lenta que el método anterior que consistía en restar las posiciones. Sin embargo, tiene otras ventajas: la podemos emplear para encontrar otros objetos que pueden cambiar de posición dentro de la matriz.

Si se quiere disponer de un medio de exploración de alcance limitado se puede restringir la exploración de la matriz a una determinada distancia de barrido (SD) por cada lado de la posición que se ocupa. Debe evitarse el salir de la matriz. Empleando valores más o menos grandes para SD, se puede disponer de un medio de exploración de corto o largo alcance. XS es la posición inicial del eje X y XE la de final. YS e YE son las posiciones equivalentes sobre el eje Y. Se obtienen sumando y restando SD a las coordenadas de la posición ocupada por el jugador. (Diagrama de flujo 7.2).


Diagrama de flujo 7.2. Exploración
FIJ. DISTANCIA EXPLORACION
LIMITE IZQ. > 0?
RESTRINGIR LIMITE IZQ. A 1
UTILIZ. TODO LIMITE IZQ.
LIMITE DER. < X?
REDUC. LIMITE DER. A X
UTILIZ. LIMITE DERECHO TOTAL
LIMITE SUP. > 0?
LIMITARLO A 1
UTILIZ. LIMITE/SUP TOTAL
LIMITE INF. < Y?
REDUCIRLO A Y
UTILIZ. LIMITE INF. TOTAL
EXPLOR. MATRIZ
TESORO?
SI
NO
CALCUL. DISTANCIA
IMPRIM. DISTANCIA
FIN MATRIZ
NO
SI
RETURN

```
10 CLS:X=5:Y=5:SD=2
1000 IF XP-SD>0 THEN XS=XP-SD EL
SE XS=1
1010 IF XP+SD<X THEN XE=XP+SD EL
SE XE=X
1020 IF YP-SD>0 THEN YS=YP-SD EL
SE YS=1
1030 IF YP+SD<Y THEN YE=YE+SD EL
SE YE=Y
1040 FOR N=YS TO YE:FOR M=XS TO
XE:IF A(N,M)=84 THEN 1050 ELSE N
EXT M,N:RETURN
1050 TY=N:TX=M:DX=ABS(XP-TX):DY=
ABS(YP-TY):DT=DX+DY:PRINT @ 0,"D
ISTANCIA AL TESORO";DT:RETURN
```

El punto desde el que se llamará a esta subrutina debe ser naturalmente el final de la línea dónde se pide al jugador que indique en qué dirección quiere moverse.

```
60 PRINT @ 200,"DIRECCION?":GOSU
B 1000
```

SUBRUTINAS DE MOVIMIENTO EN UN LABERINTO

Ya que hemos apuntado la utilidad de la rutina de exploración cuando hay cosas que se mueven dentro de la matriz, vamos a pensar en qué es lo que va a estar moviéndose y por qué. Los objetos móviles que con mayor frecuencia se mueven por los laberintos pertenecen a la extraña clase de los seres hostiles y parecen recorrer el laberinto con movimientos aleatorios. Definiremos la posición inicial ocupada por este monstruo como A(YM, XM) = 77 de forma que el monstruo se representará mediante la letra M (Diagrama de flujo 7,3).

```
30 TY=RND(Y):TX=RND(X):A(TY,TX)=
84:XM=RND(X):YM=RND(Y):A(YM,XM)=
77
```

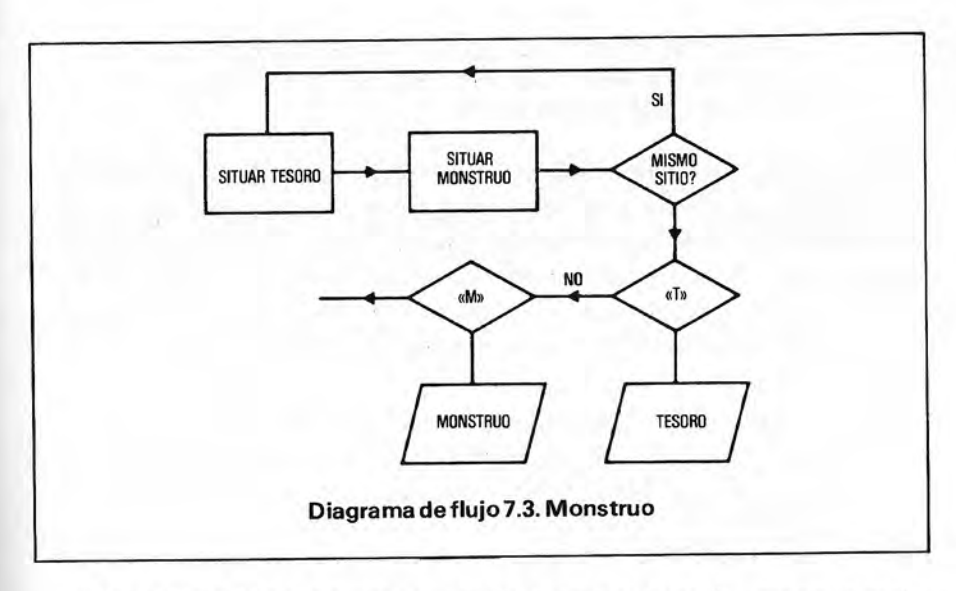


Diagrama de flujo 7.3. Monstruo

Tal y como están las cosas podría suceder que el monstruo ocupara, en un momento dado, la posición del tesoro, borrándolo. Quizás deberíamos comprobar que ambos ocupan posiciones distintas. En el caso de que ocuparan la misma, recalcularíamos sus nuevas posiciones.

```
30 TY=RND(Y):TX=RND(X):A(TY,TX)=
84:XM=RND(X):YM=RND(Y):A(YM,XM)=
77:IF TX=XM AND TY=YM THEN RUN
```

Ahora tenemos que modificar la rutina original de comprobación del tesoro para que también pueda decirnos si nosotros ocupamos la misma posición del monstruo —código 77— lo cual no sería una noticia muy agradable.

```
120 IF A(YP,XP)=84 THEN 130 ELSE
 IF A(YP,XP)=77 THEN 140 ELSE PR
INT @ 270,"ERROR":SOUND1,5:CLS:G
OTO 60
```

Añadiremos esta sencilla rutina que se encarga de enfrentarse con las consecuencias de encontrarse al monstruo.

```
140 CLS:PRINT @ 225,"MALA SUERTE
",,"TE ACABA DE MATAR EL MONSTRU
O!":SOUND 1,50:RUN
```

Ahora que ya tenemos nuestro monstruo y un método para detectar su presencia, vamos a añadir otra subrutina que permita al monstruo quedarse parado dónde estaba situado (YM, XM) o bien desplazarse de una en una posición, en cualquier dirección, hasta alcanzar una nueva posición que llamaremos MY, MX. Esto lo conseguiremos añadiendo (RND(3) — 2) a la posición anterior.

```
2000 MX=XM+(RND(3)-2):MY=YM+(RND
(3)-2):IF MX<1 THEN MX=1 ELSE IF
 MX>X THEN MX=X
2010 IF MY<1 THEN MY=1 ELSE IF M
Y>Y THEN MY=Y
```

Aunque el jugador sólo puede moverse hacia el Norte, el Sur, el Este o el Oeste, el monstruo, en cambio, puede seguir ocho direcciones distintas ya que sus coordenadas X e Y pueden cambiarse en un solo movimiento.

Cambio de la X	**Cambio de la Y**	**Resultado**
0	0	No hay movimiento
0	−1	Norte
0	+1	Sur
+1	0	Este
−1	0	Oeste
+1	−1	Noreste
−1	−1	Noroeste
+1	+1	Sureste
−1	+1	Suroeste

Si no vamos con cuidado, llenaremos la matriz de monstruos (en cuanto se les da la oportunidad, se reproducen como conejos). Debemos borrar el monstruo que ocupa la anterior posición antes de intercambiar los valores de las variables que guardan la anterior y la nueva posición. Luego colocaremos el monstruo en su nueva posición.

```
2020 A(YM,XM)=0:YM=MY:XM=MX:A(YM
,XM)=77:RETURN
```

Para permitir al monstruo que se mueva durante el juego, tenemos que añadir GOSUB 2000. Si colocamos esta instrucción al final de la rutina de comprobación de teclas, el monstruo sólo se moverá cuando nos hayamos movido nosotros.

```
70 A$=INKEY$:IF A$="" THEN 60 EL
SE A=ASC(A$):GOSUB 2000
```

Por otra parte, si llamamos a esta subrutina al principio de la línea que comprueba los valores de las teclas, el monstruo no parará de ir arriba y abajo independientemente de si nosotros nos movemos o no.

```
70 GOSUB 2000:A$=INKEY$:IF A$=""
 THEN 60 ELSE A=ASC(A$)
```

Si no se cree eso de que el monstruo se está moviendo, se puede echar una mirada al laberinto si se quiere. No hemos representado el laberinto mediante una matriz de cadenas. Por tanto, no se puede imprimir directamente las representaciones en forma de caracteres de los números contenidos en cada elemento identificado por los ejes Y y X puesto que los lugares vacíos se representan por un 0 y con este número no obtendríamos ningún carácter. Lo mejor que se puede hacer es explorar toda la matriz, sumar 128 a todos los ceros y luego imprimir la representación en caracteres de todos los elementos de la matriz. Así los espacios vacíos quedarán representados por un cuadrado negro y los otros objetos por las letras correspondientes.

```
61 PRINT @ 288,"";:FOR N=1 TO X:
FOR M=1 TO Y:CH=A(N,M):IF CH=0 T
HEN CH=128:PRINT CHR$(CH);:NEXT
M:PRINT:NEXT N:ELSE PRINT CHR$(C
H);:NEXT M:PRINT:NEXT N
```

Si se observa durante un rato el juego se comprobará que el monstruo jamás pasa por la posición del tesoro pero, en cambio, sí

que pasa por las posiciones en las que hay comida. Para hacer al·jugador la vida más difícil, además, el monstruo engulle la comida que encuentra.

Parece que sólo nos hemos olvidado de una cosa. ¿Qué pasa si el monstruo aterriza sobre nosotros o sobre el tesoro? Si lo piensa un poco comprobará que al jugador no le pasa nada en tanto en cuanto no hay en la matriz ningún número que le represente. El jugador está representado por sus coordenadas (guardadas en dos variables simples). Si el monstruo se mueve antes de que el jugador pulse una tecla, no se dará cuenta de su presencia ya que no le puede ver al jugador más que a través de la línea 120. Por otra parte, si el jugador no aprieta la tecla en el momento adecuado y el monstruo está en la posición escogida por el propio jugador, le detectará. El monstruo borrará el tesoro a menos que evitemos este problema impidiendo que se mueva si la posición a la que se quiere dirigir es la nueva que ocupa el tesoro. Para ello miraremos si en esa posible posición se encuentra el código correspondiente a T.

```
2020 IF A(MY,MX)=84 THEN 60 ELSE
A(YM,XM)=0:YM=MY:XM=MX:A(YM,XM)=
77:RETURN
```

Si al jugador le gusta arriesgarse puede hacer que aumente el peligro permitiendo que el monstruo lo detecte cuando pase cerca del mismo. Cambiése la línea a la que se salta, cuando no se ha pulsado ninguna tecla, por la línea de comparación 120 (no tiene ningún objeto pasar por las comprobaciones de las líneas 80-110 si el jugador no va a moverse).

```
70 A$=INKEY$:IF A$="" THEN 120 E
LSE A=ASC(A$)
```

SUBRUTINAS PARA MOVIMIENTOS TRIPLES

Hasta ahora hemos pintado a nuestro monstruo como un ser muy poco inteligente que vaga sin dirección precisa por el laberinto y que sólo se puede convertir en un problema si, casualmente, choca con nostros. Naturalmente pueden existir criaturas mucho más per-

versas que sigan la pista al jugador y le ataquen. Funcionan utilizando unos cálculos sencillos que les permiten saber cuál es la mínima distancia que les separa del jugador. El jugador tiene su propia posición A(YP,XP) y el monstruo la A(YM,XM). Si el monstruo resta YM de YP, el SGN del resultado será positivo o negativo según que el jugador se encontrara por encima o por debajo del monstruo. Si el resultado es positivo, el monstruo tiene que moverse hacia abajo (sumar 1 a YM) y si es negativo hacia arriba (restar 1 a YM). De forma análoga, puede decidir entre moverse hacia la izquierda o hacia la derecha por la simple comparación de las coordenadas X. Podemos colocar todas estas operaciones en una subrutina alternativa a la que antes ocupaba la línea 2000.

```
2000 MY=YM+SGN(YP-YM):MX=XM+SGN(
XP-XM)
```

Si el jugador no es capaz de enfrentarse con un monstruo que no sólo se mueve vertical y horizontalmente sino también en diagonal, tiene, en cambio, la posibilidad de restringir sus movimientos. Puede imponer, por ejemplo, que el monstruo sólo pueda escoger entre seguir la dirección Y o seguir la dirección X y ninguna más. Para ello, colóquese en la línea 2000 la siguiente instrucción ON-GOTO que escoge aleatoriamente un movimiento en la dirección X o en la dirección Y.

```
2000 ON (RND(2)) GOTO 2030,2040
2030 MY=YM+SGN(YP-YM):RETURN
2040 MX=XM+SGN(XP-XM):RETURN
```

LA RECOGIDA Y TRANSPORTE DE OBJETOS, EN UN JUEGO

Hasta este momento nos hemos ocupado de cómo hay que detectar los objetos distribuidos por la matriz. No hemos pensado aún en cómo recogerlos. Hay que llevar la cuenta de algunos objetos tales como la comida o el dinero. Basta sumar determinados valores a la variable correspondiente. Sin embargo, también deben ser eliminados de la matriz para evitar que, al volver a pasar la posición que anteriormente ocupaban, el jugador ya no los pueda recoger. Vamos

a aumentar el tamaño de la matriz a 10 × 10 pues se está poblando demasiado. Añadiremos una variable que se encargue de llevar la cuenta de la comida existente. La llamaremos F (código 70). Inicialmente tendrá el valor 20. Luego distribuimos 10 piezas de comida (cada una con 10 unidades de alimento) aleatoriamente por el laberinto mediante un bucle FOR-NEXT (Diagrama de flujo 7.4).

```
10 CLS:X=20:Y=20:SD=2:F=20
30 FOR N=1 TO 10:A(RND(Y),RND(X)
=70:NEXT N:TY=RND(Y):TX=RND(X):A
(TY,TX)=84:YM=RND(Y):XM=RND(X):A
(YM,XM)=77:IF TX=XM AND TY=YM TH
EN RUN
```

Obsérvese que colocamos primero la comida y luego el tesoro y el monstruo. Aunque así es posible que el jugador tenga menos de 10 piezas de comida, siempre tendrá el tesoro y el monstruo.

Hay que añadir ahora una rutina que detecte la comida, una señal que indique que la hemos encontrado y una rutina de borrado.

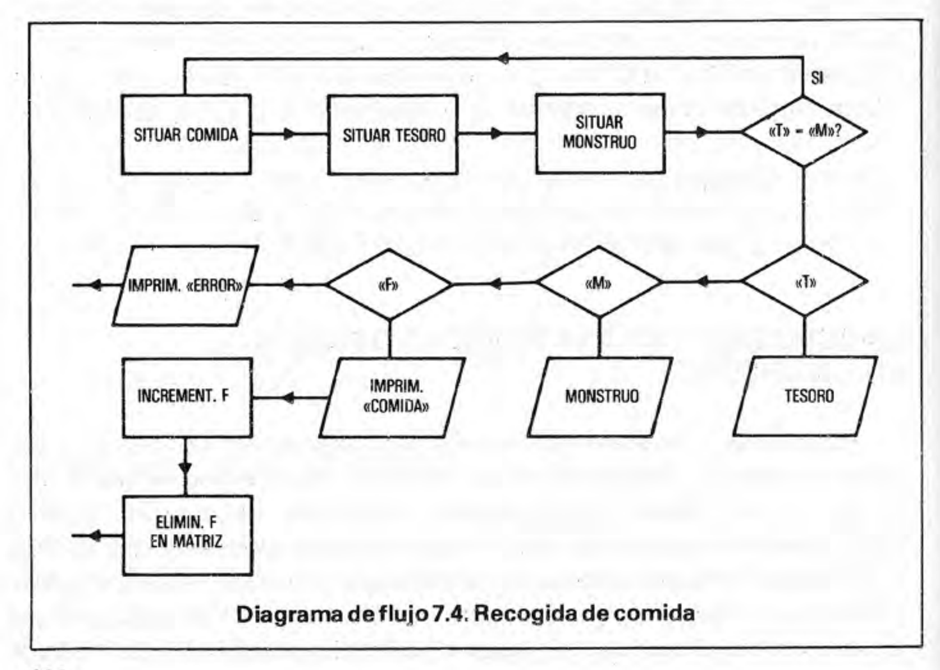


Diagrama de flujo 7.4. Recogida de comida

```
120 IF A(YP,XP)=84 THEN 130 ELSE
 IF A(YP,XP)=77 THEN 140 ELSE IF
 A(YP,XP)=70 THEN 270 ELSE PRINT
 @ 270,"ERROR":SOUND 1,5:CLS:GOT
O 60
270 PRINT @ 270,"COMIDA":F=F+10:
SOUND 100,5:CLS:A(YP,XP)=0:GOTO
60
```

Presumiblemente el jugador quiere emplear la comida durante un buen rato. Si el valor de F desciende hasta el cero, el jugador se morirá de hambre así que añadamos todo esto a la siguiente línea de comprobación y pongamos también un mensaje que se refiera a las consecuencias de la falta de alimento.

```
70 GOSUB 2000:A$=INKEY$:F=F-1:IF
 F<0 THEN 160 ELSE IF A$="" THEN
 120 ELSE A=ASC(A$)
160 CLS:PRINT @ 225,"MALA SUERTE
","ACABAS DE MORIR DE HAMBRE":SO
UND 1,50:RUN
```

Téngase en cuenta que, en este caso, se decrementa F incluso si el jugador no se ha movido. Si se quiere que las cosas sean un poco más fáciles y se desea disponer de más tiempo para pensar los propios movimientos, colóquese la asignación F = F − 1 en una posición más retrasada dentro de la misma línea. Así, sólo consumirá comida cuando se mueva.

```
70 GOSUB 2000:A$=INKEY$:IF F<0 T
HEN 160 ELSE IF A$="" THEN 120 E
LSE F=F-1:A=ASÇ(A$)
```

Se pueden colocar otros objetos (una espada S para matar monstruos, por ejemplo (código 83)) en la matriz y detectarlos de forma similar. Sería también muy interesante poder recoger estos objetos y llevarlos durante un buen rato por si hay que utilizarlos más adelante.

Vamos a colocar una espada en la matriz. Dejaremos que el juga-

dor la detecte e indicaremos que la lleva haciendo S = 1. Para ser justos vamos a colocarla antes de situar al monstruo y al tesoro. Así puede ocurrir que el jugador no disponga de ella.

```
30 FOR N=1 TO 10:A(RND(Y),RND(X)
=70:NEXT N:A(RND(Y),RND(X))=83:T
Y=RND(Y):TX=RND(X):A(TY,TX)=84:Y
M=RND(Y):XM=RND(X):A(YM,XM)=77:I
F TX=XM AND TY=YM THEN 30
120 IF A(YP,XP)=84 THEN 130 ELSE
 IF A(YP,XP)=77 THEN 140 ELSE IF
 A(YP,XP)=70 THEN 270 ELSE IF A(
YP,XP)=83 THEN 283 ELSE PRINT @
270,"ERROR":SOUND 1,5:CLS:GOTO 6
0
283 PRINT @ 270,"ESPADA":S=1:SOU
ND 25,35:GOTO 60
```

En juegos de aventuras más complejos, donde el jugador puede llevar distintas clases de objetos, resulta muy ventajoso agruparlos en una cadena ya que así es más fácil decodificar luego de qué objeto se trata. Podemos emplear C$ para llevar nuestros objetos como su representación CHR$ de los números de la matriz. Iremos añadiendo objetos al final de C$ según los vayamos recogiendo.

```
283 C$=C$+CHR$(83):PRINT @ 270,"
ESPADA":SOUND 25,35:GOTO 60
```

C$ puede tener una longitud de hasta 255 caracteres y, como no borramos de la matriz la espada cuando la tomamos, podríamos llegar a llevar 255 espadas aunque no hay ningún motivo para ello, por el momento, puesto que así tampoco aumentan nuestras posibilidades de sobrevivir.

También se puede colocar la palabra «espada» en la cadena C$ y luego imprimir C$ aunque esto acarrea complicaciones si se lleva más de un objeto o queremos dejar el que llevamos.

```
283 C$=C$+"ESPADA":PRINT @ 270,C
$:SOUND 25,35:GOTO 60
```

Si vamos a tener muchos objetos diferentes y empleamos una sentencia IF-THEN para verificar cada uno de ellos, la línea 120 se va a complicar bastante. Por tanto, vamos a replantearnos este punto de salto del programa. El resultado que tenemos mayor posibilidad de obtener, al realizar toda esa serie de comprobaciones, es el de que la posición A(YP,XP) esté vacía y, por tanto, el resultado será 0. Empecemos, pues, por retirar esta comprobación. Si recurrimos a las letras A a Z para representar los distintos tipos de objetos (el tesoro, los monstruos, la comida y otras cosas) podemos llegar a tener un total de 26 cosas distintas. Podemos separarlas fácilmente mediante una instrucción ON-GOTO (A(YP,XP) — 64) ya que esas letras tienen por código ASCII los que van del 65 al 90. Si primero no nos encargamos del 0 generaremos un error FC ya que se producirá un número negativo e incorrecto. Los números de línea que lógicamente deberían utilizarse son aquellos que sean 200 unidades mayores que el correspondiente código ASCII. Como el BASIC del Dragón no permite alterar el orden de las líneas, el jugador tendría que volver a escribir unas cuantas si no hubiéramos empleado unos números tan extraños la primera vez.

```
120 IF A(YP,XP)=0 THEN PRINT @ 2
70,"ERROR":SOUND 1,5:CLS:GOTO 60
ELSE ON(A(YP,XP)-64)GOTO 265,266
,267,268,269,270,271,272,273,274
,275,276,277,278,279,280,281,282
,283,284,285,286,287,288,289,290
270 PRINT @ 270,"COMIDA":F=F+10:
SOUND 100,5:CLS:A(YP,XP)=0:GOTO
60
283 C$=C$+CHR$(83):PRINT @ 270,"
ESPADA":SOUND 25,35:GOTO 60
```

El jugador va a tener que añadir las líneas apropiadas para las letras con las que pueda encontrarse dentro del laberinto. Lo más sencillo es utilizar un salto a la línea que originalmente se encargaba de las consecuencias. Por ejemplo, para el monstruo (código 77) las líneas adecuadas son la 277 y la 140.

```
277 GOTO 140
```

EL REPARTO DE POSIBILIDADES ENTRE JUGADORES

Si lo único que tuviera que hacer el jugador fuera comprobar si ha cogido una espada, todo sería fácil, especialmente si siguiera la sugerencia acerca de la línea 283 en la que se señalaba que cuando el jugador lleva la espada, S vale 1. En este caso, tendría que introducir en la línea 140 una comprobación del valor de S. Para darle al monstruo alguna posibilidad, haremos que el jugador tenga un 50 % de posibilidades de ganar aún contando con la espada. Esto lo conseguiremos mediante la «variable de la suerte» (LU).

```
140 CLS:LU=RND(2):IF S=0 OR LU=1
 THEN PRINT @ 225,"MALA SUERTE",
,"TE ACABA DE COMER EL MONSTRUO!
":SOUND 1,50:RUN:ELSE PRINT @ 22
5,"HAS TENIDO SUERTE","HAS MATAD
O AL MONSTRUO":SOUND 120,40:GOTO
 60
```

Si el jugador lleva varias cosas en la cadena C$ puede utilizarlas mediante la función INSTR que le dará el valor 0 si C$ no contiene el carácter que él quiere.

```
140 CLS:LU=RND(2):IF INSTR(1,C$,
"S")=0 OR LU=1 THEN PRINT @ 225,
"MALA SUERTE",,"TE ACABA DE MATA
R EL MONSTRUO!":SOUND 1,50:RUN:E
LSE PRINT @ 225,"HAS TENIDO SUER
TE","HAS MATADO AL MONSTRUO":SOU
ND 120,40:GOTO 60
```

EL CALCULO DEL TIEMPO EN LAS OPERACIONES DE UN JUEGO

No hay nada que dure indefinidamente pero algunas cosas son más perecederas que otras. ¿Qué le parece si introducimos factores que controlen su desgaste? (Diagrama de flujo 7.5).

Si guardamos el valor del temporizador TIMER en una variable

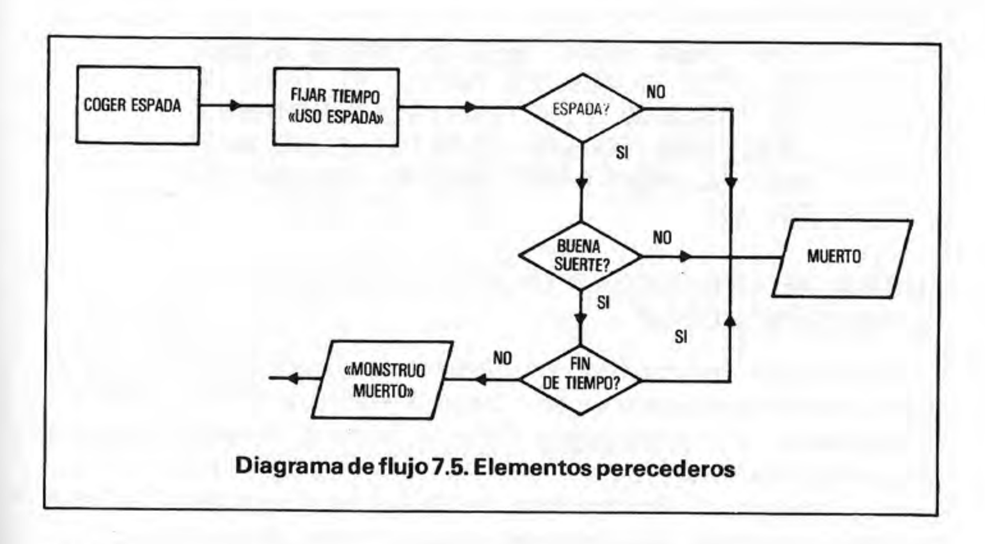


Diagrama de flujo 7.5. Elementos perecederos

(ST, tiempo de la espada) cuando la recojamos y consultemos ese valor en el momento en que queramos utilizar el arma, mediante (TIMER/50) — ST podemos determinar cuánto tiempo llevamos con la espada en la mano. Podemos también imponer un límite máximo de duración del objeto (dos minutos en este ejemplo). Si se escogiera la solución más normal —inicializar el temporizador cada vez que recogiera un objeto— sólo podría «envejecer» un objeto cada vez. El temporizador cuenta de 0 a 64 K con una frecuencia de 50 recuentos por segundo: se dispone, por tanto, de 25 minutos de juego.

```
283 C$=C$+CHR$(83):PRINT @ 270,"
ESPADA":SOUND 25,35:ST=TIMER/50:
GOTO 60
140 CLS:LU=RND(2):IF S=0 OR LU=1
OR (TIMER/50)-DT>120 THEN PRINT
@ 225,"MALA SUERTE",,"TE ACABA D
E DEVORAR EL","MONSTRUO!":SOUND
1,50:RUN:ELSE PRINT @ 225,"HAS T
ENIDO SUERTE","HAS MATADO AL MON
STRUO":SOUND 120,40:GOTO 60
```

o

```
140 CLS:LU=RND(2):IF INSTR(1,C$,
"S")=0 OR LU=1 OR (TIMER/50)-ST>
```

```
120 THEN PRINT @ 225,"MALA SUERT
E",,"TE ACABA DE MATAR EL MONSTR
UO!":SOUND 1,50:RUN:ELSE PRINT @
225,"HAS TENIDO SUERTE","HAS MAT
ADO AL MONSTRUO":SOUND 120,40:GO
TO 60
```

FORMA DE ABANDONAR OBJETOS RECOGIDOS DURANTE EL JUEGO

Si el jugador ha empleado ya un elemento perecedero debe reasignar la variable correspondiente o sacar el elemento de C$. Además, normalmente se impone algún límite al número de elementos que se pueden llevar simultáneamente. La forma de abandonar los elementos recogidos eliminándolos de C$ es muy sencilla si se han guardado dentro de C$ los elementos en forma de caracteres individuales. La función INSTR devuelve la posición (CP) de una cadena dada dentro de otra. Esa primera cadena se puede definir cada vez que sea necesario. Se puede sacar de C$ un objeto determinado separando una sección de la cadena por cada lado del objeto que queremos sacar (Fig. 7.2) y, después, volviendo a juntar las dos partes resultantes para obtener la nueva C$. Se puede emplear esta misma subrutina para abandonar un objeto o bien para perderlo si hemos agotado su tiempo de utilización. Por tanto, la escribiremos como una subrutina independiente. Si ahora le damos a la cadena que buscábamos dentro de C$ el valor del carácter que estamos buscando y llamamos a la subrutina desde el final de la línea 140, perderemos automáticamente nuestra espada (si hemos escapado del monstruo).

```
140 CLS:LU=RND(2):IF INSTR(1,C$,
"S")=0 OR LU=1 OR (TIMER/50)-ST>
120 THEN PRINT @ 225,"MALA SUERT
E",,"TE ACABA DE MATAR EL MONSTR
UO!":SOUND 1,50:RUN:ELSE PRINT @
 225,"HAS TENIDO SUERTE","HAS MA
TADO AL MONSTRUO":SOUND 120,40:T
S="S":GOSUB 3000:GOTO 60
3000 CP=INSTR(1,C$,T$):C$=LEFT$(
C$,CP-1)+RIGHT$(C$,CP+1):RETURN
```

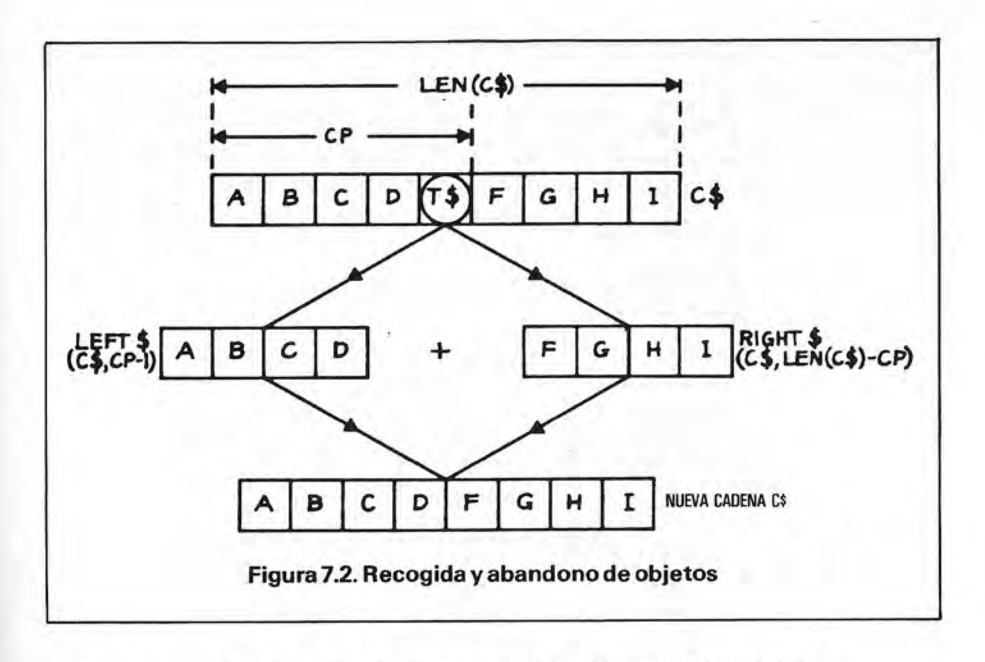


Figura 7.2. Recogida y abandono de objetos

SUBRUTINAS PARA LA ELECCION ENTRE DIVERSAS ALTERNATIVAS

Podemos añadir la posibilidad de escoger o no un objeto llamando a una subrutina de elección (situada en la línea 4000) desde la línea correspondiente al objeto (por ejemplo la 283) (Diagrama de flujo 7.6). Así se recibe un informe acerca de qué objetos estamos llevando en un momento dado. La información consiste en la impresión de las iniciales de los objetos que llevamos. Luego, se nos pregunta si queremos emplear alguno de ellos.

Si no queremos, el programa retrocede a la comprobación de teclas. En el caso de que sí queramos utilizarlo, se comprueba si disponemos de suficiente espacio para llevarlo. Se puede inicializar al principio del programa el límite de objetos que se pueden llevar (CL) en C$. Luego se puede comprobar fácilmente si lo rebasamos comparando CL con la longitud de C$ (LEN(C$)) cada vez que nos encontremos con un nuevo objeto en el laberinto.

Si hay suficiente espacio, se nos comunicará y podremos continuar. Si no lo hay, se nos preguntará si deseamos abandonar alguno de los objetos que llevamos. Si no queremos, el programa salta di-

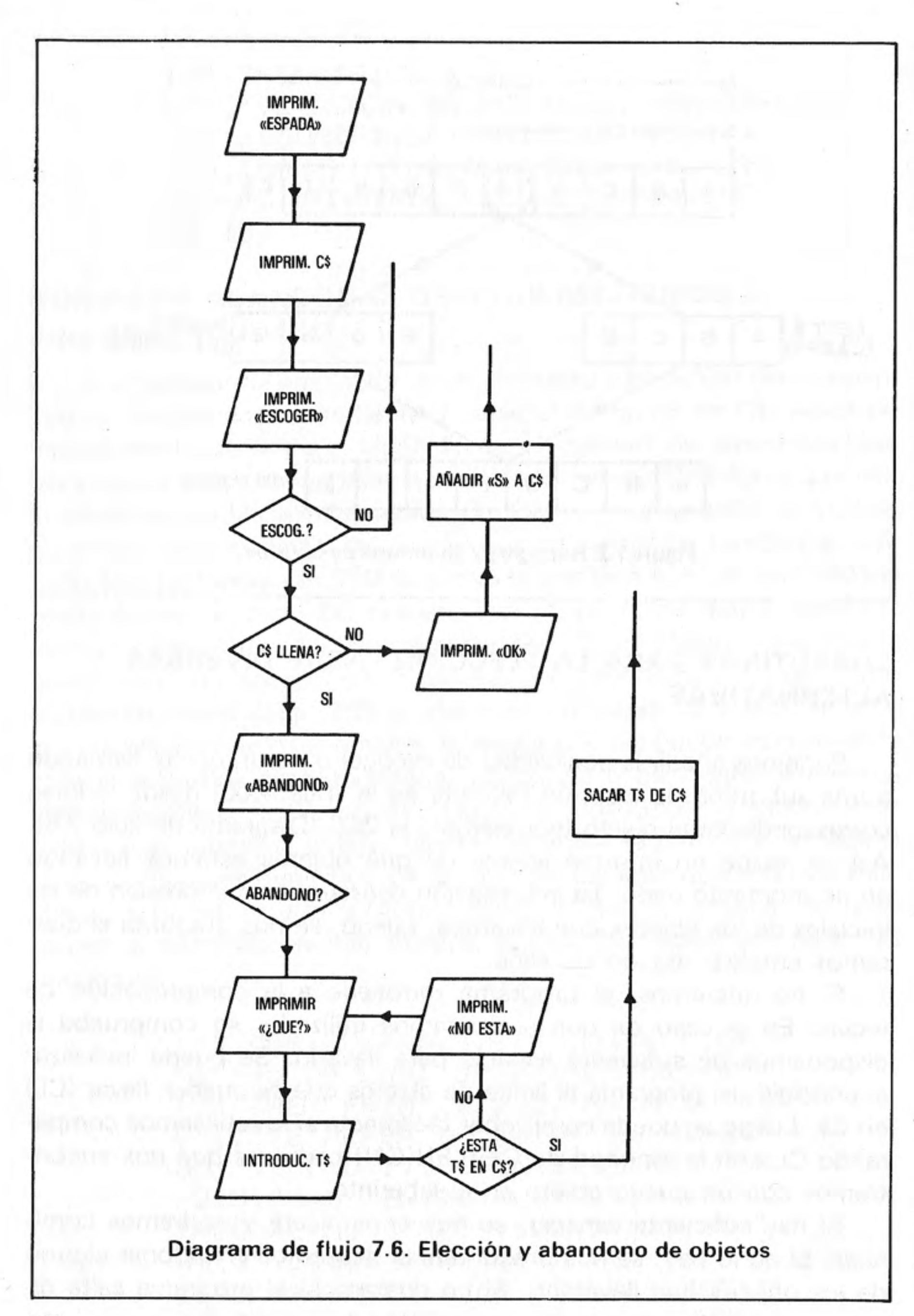


Diagrama de flujo 7.6. Elección y abandono de objetos

rectamente a la comprobación de teclas. Si deseamos abandonar algo se nos preguntará qué y, en caso de que no llevemos el objeto que deseamos abandonar, se nos indica así y se nos da otra oportunidad. Si realmente lo llevamos, el objeto será eliminado de C$ por la subrutina de la línea 3000 que ya hemos descrito y se añadirá el nuevo objeto encontrado a la cadena C$.

```
10 CLS:X=20:Y=20:SD=2:F=20:CL=5
283 PRINT @ 270,"ESPADA":GOSUB 4
000:C$=C$+CHR$(83):SOUND 25,35:S
T=TIMER/50:GOTO 60
4000 PRINT @ 320,"LLEVAS",C$;:PR
INT @ 352,"QUIERES COGER ALGO?",
" S/N";:INPUT Q$:IF Q$<>"S" THEN
60 ELSE IF LEN(C$)>CL THEN 4010
ELSE PRINT "OK":RETURN
4010 PRINT @ 384,"TIENES LAS MAN
OS LLENAS","QUIERES ABANDONAR","
ALGO?(S/N)";:INPUT Q$:IF Q$<>"S"
 THEN 60 ELSE PRINT "QUE?",:INPU
T Q$:IF INSTR(1,C$,Q$)=0 THEN GO
TO 4020 ELSE GOSUB 3000:SOUND 5,
10:RETURN
4020 PRINT @ 384,"NO LLEVAS NING
UN(A)";Q$,,,,:SOUND 3,10:GOTO 40
10
```

Si se quiere recibir una información más completa que la simple impresión de las iniciales, se puede añadir una llamada a una subrutina que se encargue de eso. Resulta una técnica especialmente útil para juegos complejos. Una subrutina así separará de C$ un carácter cada vez y utilizará su valor ASCII en una instrucción ON-GOSUB similar a la de la línea 120. Esta instrucción saltará a una subrutina que describa totalmente los objetos correspondientes a las iniciales. Las descripciones particulares se las dejamos a su libre elección.

LA SELECCION DE ASIGNACIONES A DIFERENTES ATRIBUTOS

Hasta este momento hemos estado recorriendo el laberinto y librando tremendas batallas enfrentándonos con los terrores que se

cernían en las esquinas más insospechadas. Sólo nos ha ayudado la suerte (o nos ha perjudicado) de forma aleatoria. Podría ser una buena idea imprimir nuestra propia personalidad al sistema asignándonos nuestras propias fortalezas y debilidades en distintos campos. Esta asignación podría resultar muy importante para nuestro éxito aunque, en algunos casos, uno puede ser el substituto adecuado de las otras. La fuerza es una ventaja importante en combate, pero con una mayor inteligencia, se podría haber evitado la batalla y con una mejor agilidad podríamos habernos escapado para combatir en una ocasión más propicia. Desde el principio del programa se podría saltar a una rutina de selección que permitiera distribuir una cierta cantidad total de puntos (PT) de forma equitativa entre tres atributos: agilidad (PA), inteligencia (PI) y fuerza (PS) que deben sobrepasar o ser iguales a una cantidad mínima (PM) (Diagrama de flujo 7.7).

Se han incluido instrucciones para comprobar que todas las cantidades entradas por el jugador rebasan el mínimo establecido. También se comprueba el valor total para evitar que el jugador haga trampas convirtiéndose en un superhéroe. A esto siguen las instrucciones para el jugador, animadas con un poco de sonido y con destellos en la pantalla. La otra pantalla de texto necesaria para conseguir esto se obtiene haciendo SCREEN 0,1 al final del PRINT. Se impide que durante un tiempo vuelva a la pantalla por defecto SCREEN 1,1 mediante una serie de sonidos de retardo.

```
10000 PT=12:PM=1:CLS:PRINT"SELEC
CION DEL PERFIL DE PERSONALIDAD"
,,"TIENES";PT;"PUNTOS","PARA DIS
TRIBUIR ENTRE",PRINT @ 225,"AGIL
IDAD";:PRINT @ 257,"INTELIGENCIA
";:PRINT @ 289,"FUERZA";:PRINT @
 448,"TIENES QUE TENER AL MENOS"
;PM;"UNIDADES DE CADA TIPO"
10010 PRINT @ 245,"";:INPUT PA:I
F PA<PM THEN 10030 ELSE PRINT @
277,"";:INPUT PI:IF PI<PM THEN 1
0040 ELSE PRINT @ 309,"";:INPUT
PS:IF PS<PM THEN 10050
10020 IF PA+PI+PS=<PT THEN RETUR
N ELSE FOR N=1 TO 10:PRINT @ 352
```

```
,"TRAMPOSO!!!":SCREEN 0,1:SOUND
5,N:NEXT N:GOTO 10000
10030 PRINT @ 352,"NO ERES LO SU
FICIENTEMENTE AGIL PARA MOVERTE-
PIENSA OTRA VEZ":SCREEN 0,1:SOUN
D 5,25:GOTO 10000
10040 PRINT @ 352,"COMO VAS A PE
NSAR EN QUE HACER AHORA?":SCREEN
 0,1:PLAY"ABABABABAB":GOTO 10000
10050 PRINT @ 352,"PULSAR ESA TE
CLA TE HA DEJADO AGOTADO-PIENSA
OTRA VEZ"!;:SCREEN 0,1:FOR N=255
 TO 1 STEP -10:SOUND N,1:NEXT N:
GOTO 10000
```

El efecto exacto de los distintos atributos, cómo ayudan o perjudican al jugador, depende de las consecuencias de sus acciones y a ellas nos referiremos más adelante.

COMO SE GUARDA LA POSICION DEL JUGADOR AL ABANDONAR EL JUEGO

Una característica esencial de los juegos de cierta complejidad es la posibilidad que ofrecen de abandonarlos a medias cuando hay que irse al trabajo, a comer o a dormir. No cuesta mucho disponer de esta posibilidad siempre que el jugador no altere el orden en que carga o almacena sus programas. El Dragón no guarda las variables junto a los programas pero permite hacerlo separadamente guardándolas como ficheros de datos («DATA FILES»).

Añadiremos una subrutina en la línea 60000 que almacene la posición que el jugador ocupaba cuando abandonó el juego. La primera cosa que hay que hacer es comunicarle al sistema que queremos abrir un fichero (operación OPEN) de datos para salida. Hay que darle un nombre a ese fichero. No debe ser de más de ocho letras (al igual que ocurre con los nombres de los programas). Los caracteres posteriores al octavo en nombres de longitud mayor se ignoran Aún así, los nombres largos acostumbran a causar confusión cuando

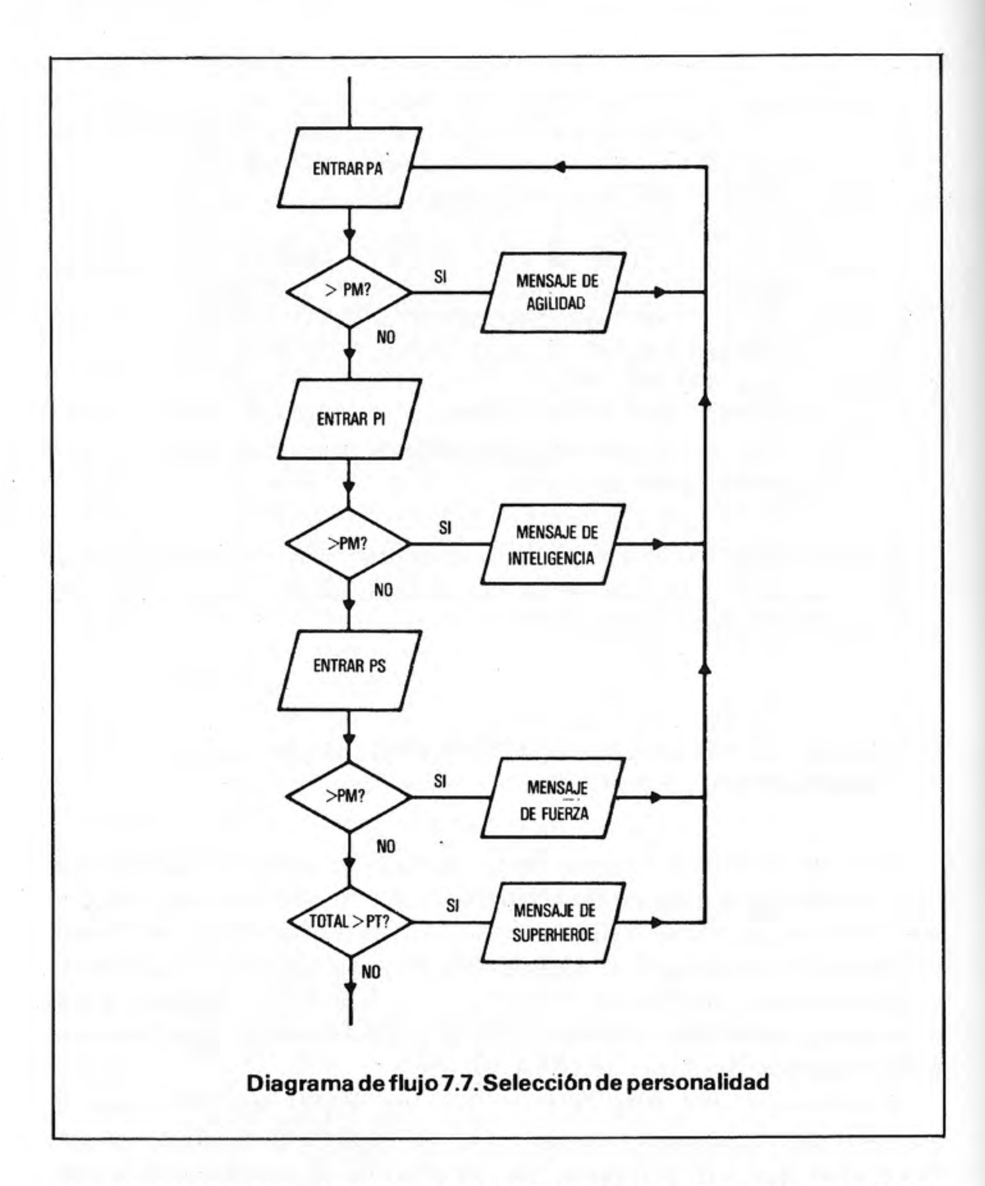


Diagrama de flujo 7.7. Selección de personalidad

son parecidos (por ejemplo, DRAGONLOCO o DRAGONLANDIA). Por tanto, añadiremos instrucciones que comprueben la validez del nombre que deseamos darle al fichero. Se comprobará la longitud del nombre y si el valor ASCII del primer carácter corresponde a una letra mayúscula (ASCII 65-90) (Diagrama de flujo 7.8).

```
60000 CLS4:PRINT @ 65,"NOMBRE DE
L FICHERO A GUARDAR";:INPUT N$:I
F LEN(N$)>8 THEN 60000 ELSE IF A
SC(LEFT$(N$,1))<65 OR ASC(LEFT$(
N$,1))>90THEN 60000 ELSE PRINT @
 128,"CUANDO CASSETTE LISTO PULS
AR UNA TECLA":MOTOR ON:AUDIO ON:
INPUT Q$:OPEN"O",#-1,N$
```

Se recuerda que se debe poner en marcha el cassette. Téngase en cuenta que el control remoto del cassette se desactiva mediante MOTOR ON para que se pueda posicionar la cinta.

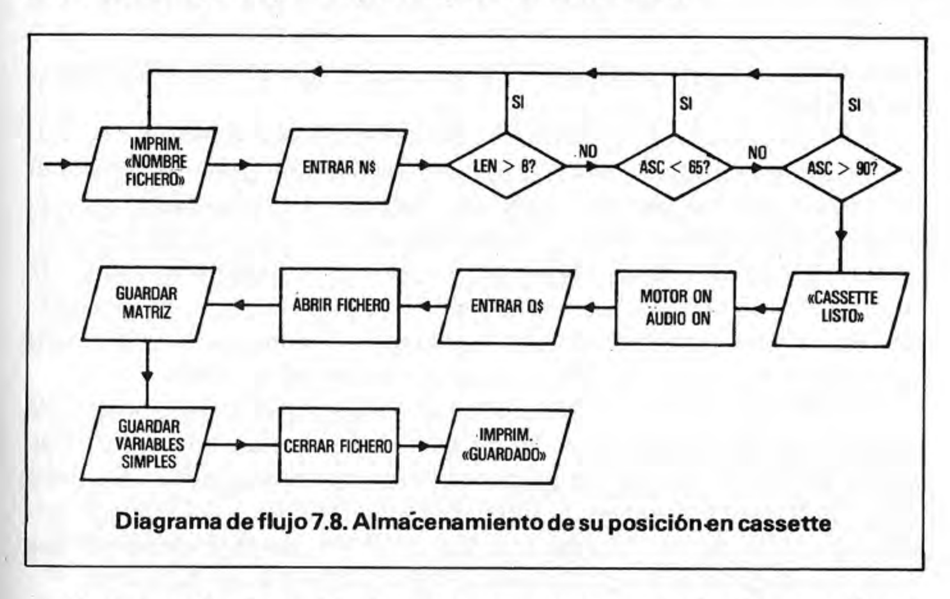


Diagrama de flujo 7.8. Almacenamiento de su posición en cassette

Para guardar la matriz tenemos que recorrer todas las coordenadas secuencialmente e imprimir cada elemento en el cassette al que el Dragón ve como el dispositivo número 1 (# —1).

```
60010 FOR N1=1 TO Y:FOR N2=1 TO
X:PRINT #-1,A(N1,N2):NEXT N2,N1
```

De forma análoga, guardamos todas las variables simples que tenemos que almacenar y luego cerramos el fichero (operación CLOSE)

e imprimimos un mensaje para indicar que el fichero ha sido almacenado.

```
60020 PRINT#-1,YP,XP,TY,TX,MY,MX
,F,C$,CL:CLOSE#-1:PRINT @ 490,"F
ICHERO GUARDADO";:RETURN
```

Para poder parar y guardar o almacenar el programa, en cualquier momento, se puede emplear una comprobación para una tecla en particular que le conduzca a la subrutina de almacenamiento en el programa principal. Otra solución más sencilla consiste en pulsar la tecla BREAK y escribir seguidamente GOTO 60000 como una instrucción directa. En este último caso resulta esencial NO hacer RUN 60000 si no quiere que lo que está a punto de guardar se borre. Para asegurarse de que se ha almacenado el fichero puede rebobinar y hacer SKIPF.

Para empezar el juego otra vez se necesita, por descontado, volver a cargar el programa del juego en BASIC. Se debe dimensionar la matriz antes de leer los datos del cassette, asegurándose de que los lee en el orden correcto. Colocaremos una rutina de re-carga en la línea 61000 de forma que usted pueda volver a empezar mediante RUN 61000 siempre que antes haya colocado la cinta en su posición Se le pedirá el nombre del fichero y luego se dimensionará la matriz principal y empezará la rutina de carga propiamente dicha.

No se ha previsto la comprobación de la validez del nombre de fichero que se pueda entrar. Si no está, no se podrá encontrar. Pulsando ENTER se cargará el primer fichero que se encuentre. Se debe abrir el dispositivo número 1 para entrada (OPEN # —1) y luego entrar los datos desde la cinta con INPUT #—1 asegurándose de que no se ha llegado al fin de fichero (EOF) (Diagrama de flujo 7.9).

```
61000 X=10:Y=10:CLS2:PRINT @ 129
,"ENTRAR NOMBRE FICHERO PARA EMP
EZAR DE NUEVO EL JUEGO":INPUT N$
:DIM A(Y,X):OPEN "I",#-1,N$:FOR
N1=1 TO Y:FOR N2=1 TO X:IF EOF(-
1)THEN 61010 ELSE INPUT #-1,A(N1
,N2):NEXT N2,N1:INPUT #-1,YP,XP,
TY,TX,MY,MX,F,C$,CL
61010 CLOSE #-1:GOTO 60
```

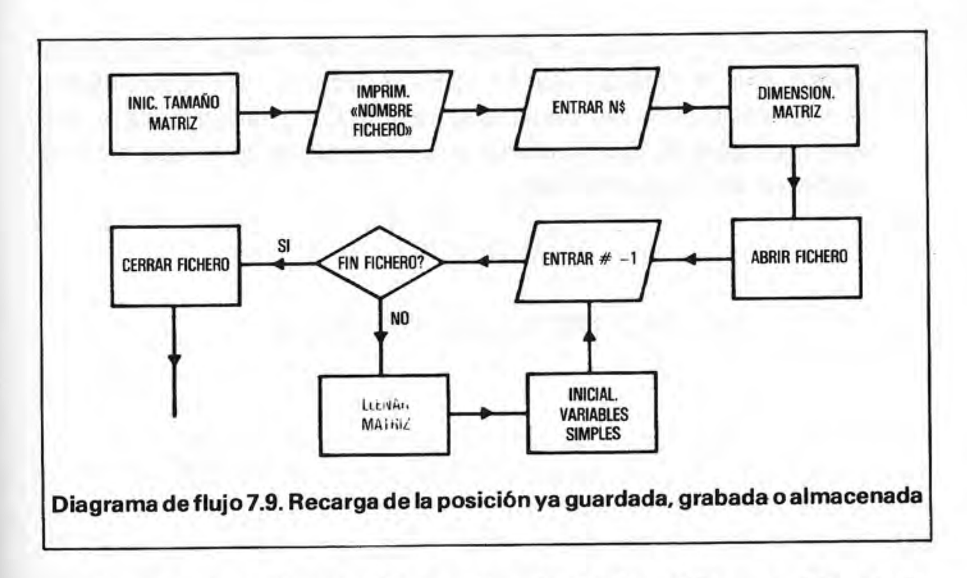


Diagrama de flujo 7.9. Recarga de la posición ya guardada, grabada o almacenada

Cuando ya se han vuelto a entrar todos los datos, se debe reiniciar el juego saltando al punto adecuado del programa ya que no se puede volver al principio del programa y hacer RUN, ya que haciendo esto todas las variables se reinicializarían dando al traste con todos nuestros esfuerzos por conservar sus antiguos valores.

AHORA ES SU TURNO

El propósito principal de este capítulo era ofrecer rutinas útiles para que el lector las pudiera aplicar en juegos en los que emplearan laberitos. Sería una buena idea, por tanto, que adquiriera un poco de práctica en su uso y modificación.

1) Escriba unas rutinas de exploración y detección para la «máquina de hacer laberintos» que funcionen en tres dimensiones.
2) Coloque monstruos en un laberinto tridimensional pero restrinja sus movimientos a un solo nivel.
3) Limite los movimientos de los monstruos de forma que sólo se mueva hacia usted si se encuentra a una determinada distancia.
4) Escriba una rutina que almacene y cargue juegos para la «ventana en el laberinto».

5) Relacione el número de objetos que puede llevar simultáneamente con la fuerza que ha especificado en su personalidad.
6) Escriba otra línea de saltos que divida C$ y luego salte a una subrutina que dé descripciones más completas de lo que está llevando en un momento dado.

Capítulo VIII

GRAFICOS Y SONIDOS

Las consecuencias que se deriven de sus acciones pueden significar la diferencia entre un juego bueno y otro malo. Son por tanto, una parte muy importante en el diseño de programas de juego. Resulta obvio, como usted ya habrá notado, que la consecuencia inmediata más sencilla es que usted gane o pierda. Como este esquema aburre en seguida, hemos añadido mecanismos que llevan la cuenta de la puntuación o bien le hemos dado al jugador la oportunidad de tener varias vidas. Otra idea a la que hemos recurrido ha sido la que nos permitía seguir jugando mientras los daños sufridos no rebasaran un cierto límite. También ocurre a menudo que nos quedamos sin comida o que se no acaba el tiempo. El paso siguiente consistió en introducir la idea de una suerte aleatoria que podía jugar en favor o en contra nuestra pero que era imposible vaticinar. En el capítulo anterior explicamos cómo se podía adquirir y utilizar elementos de ayuda que le podían ser útiles más adelante (espada, por ejemplo). Finalmente explicamos cómo preparar un perfil de personalidad.

Vamos a echar ahora una ojeada a la forma de conseguir rutinas más interesantes que se encarguen de las consecuencias de los actos que se han desarrollado anteriormente. Si se examina de nuevo el programa en el que la pantalla se desplazaba («A toda marcha») se comprobará que las consecuencias de salirse de la carretera o de chocar contra un obstáculo amarillo eran diferentes pero tan sólo en la importancia del daño recibido. Al final del capítulo le sugerimos que colocara bloques de distintos colores en la carretera y escribiera rutinas que detectaran con qué tipo de obstáculo se había chocado. Si no consiguió diseñar esas rutinas, ahí va una solución (Diagrama de flujo 8.1).

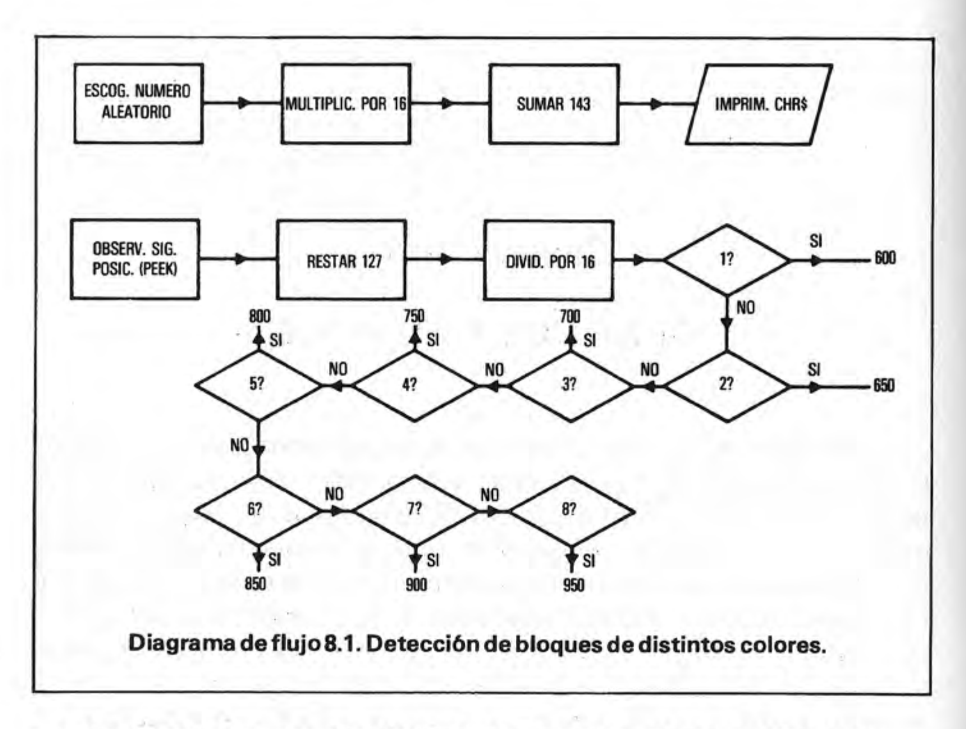


Diagrama de flujo 8.1. Detección de bloques de distintos colores.

LA OBTENCION DE BLOQUES DE DIFERENTES COLORES

Se pueden obtener obstáculos de todos los colores con (((RND(8) — 1) *16) + 143))). Por tanto, basta con que se cambie el CHR$(159) de la línea 110 por esta expresión. Para detectar todos los bloques separadamente se pueden substituir las líneas 210 y 220 por una instrucción ON GOTO que tome sus códigos ASCII y salte a distintas consecuencias siempre que el jugador se encargue, en primer lugar, de CHR$(96). Todas las rutinas referidas a los colores deben terminar con un GOTO 230 para que vuelvan a la comprobación de daños sufridos y al resto del programa.

```
110 D=RND(11)-1:C=D+A:PRINT @ (4
80)+C,(((RND(8)-1)*16)+143);:PRI
NT @ (192+LP),CHR$(128);:PRINT @
 (224+E),"u";:PRINT @ 52,"T
  D";:PRINT @ 53,USING "####";TI
;:PRINT @ 59,USING "####";DI;:PR
INT @ 511,"";:PLAY "T255;O4;B"
```

```
210 IF H=96 THEN 220 ELSE ON ((H
-127)/16) GOTO 600,650,700,750,8
00,850,900,950
220 DA=DA+5:SOUND 100,1
```

Lo primero que tenemos que hacer es colocar de nuevo el resultado de chocar contra un obstáculo amarillo ya que hemos borrado la línea 210 original. El amarillo es el segundo color del Dragón, por tanto, necesita estar en la línea 650. También añadiremos ahora un breve mensaje de advertencia que aparecerá en la parte superior de la pantalla, que será diferente cada vez, y que desaparecerá conforme se desplace la pantalla.

```
650 PRINT @ 0,"BUMP!";:DA=DA+5:S
OUND 100,1:GOTO 230
```

INTRODUCCION DE FACTORES DE RETRASO EN UN PROGRAMA

Como ya hemos sugerido, el tiempo puede ser un factor importante, así que ¿por qué no añadimos algunos factores de retraso? Una de las cosas más irritantes para un conductor son los semáforos. Vamos a añadir unos cuantos (indicados por un bloque rojo) que le retrasarán por un rato. Los semáforos en rojo le retrasarán, al menos, durante 500 unidades de tiempo o incluso hasta cuatro veces más. El ámbar le detendrá por un tiempo más largo pero fijo y el verde tiene un factor aleatorio para ajustarse a su suerte con las luces de tráfico (Diagrama 8.2).

```
750 PRINT @ 0,"SEMAFOROS":PRINT"
ROJO":FOR N=1 TO RND(2000)+500:N
EXT N:PRINT @ 32,"AMBAR":FOR N=1
TO 500:NEXT N:PRINT @ 32,"VERDE"
:FOR N=1 TO RND(200)+50:NEXT N:G
OTO 230
```

LA INTENSIFICACION DE LOS SONIDOS

Si el jugador tiene la mala suerte de chocar contra un bloque azul vendrá la policía, haciendo sonar incluso la sirena (Diagrama de

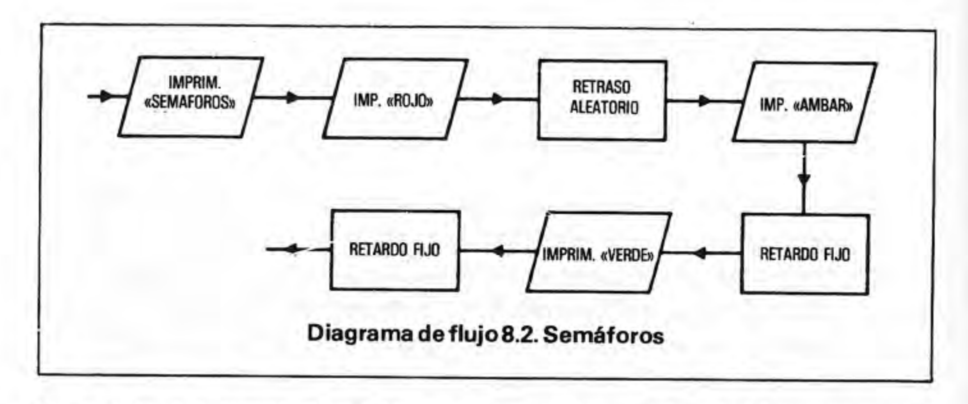


Diagrama de flujo 8.2. Semáforos

flujo 8.3). La sirena se hace cada vez más y más rápida, ya que el volumen se va doblando en cada repetición mediante V > . Debe asegurarse de no obtener un sonido demasiado alto (V > 31). Este es un punto ideal para colocar unos cuantos gráficos simples mientras la policía le hace la prueba del alcohol. Mientras está soplando podrá ver cómo aparece un histograma en la parte superior de la pantalla y se preguntará angustiado si el color verde alcanzará o no el límite fatal de los 80 mg que tan funestas consecuencias puede tener en su futura carrera automovilística. Con un nivel bajo de alcohol se recibe un mensaje de tono bastante moderado; un nivel algo mayor lleva asociada una recomendación más enérgica y si el nivel rebasa los 80 mg... ¿qué es lo que usted espera?

```
700 PRINT @ 0,"ACABAS DE ATROPEL
LAR A ALGUIEN","VIENE LA POLICIA
";:PLAY"V1L2T4FGV>FGV>FGV>FGV>FG
V31FG"
710 PRINT @ 0,"PERDON, SE/OR":PR
INT @ 32,"SOPLE AQUI POR FAVOR";
:SOUND 150,20:PRINT @ 15,CHR$(12
8);STRING$(16,159);:PRINT @ 25,"
80 MG";
720 LU=RND(16):FOR N=1 TO LU:PRI
NT @ (15+N),CHR$(149);:SOUND N,N
:NEXT N:IF LU>10 THEN 740 ELSE I
F LU<5 THEN SOUND 255,10:PRINT @
 0,"GRACIAS",,"SIENTO LAS MOLEST
IAS";:SOUND 150,30:GOTO 230
```

```
730 PRINT @ 32,"CREO QUE USTED H
A BEBIDO";:SOUND 1,20:PRINT @ 0,
"HA TENIDO SUERTE POR ESTA VEZ",
"PERO VAYA CON CUIDADO!!!!!";:SO
UND 40,20:GOTO 230
740 PRINT @ 32,"SE QUE HA ESTADO
 BEBIENDO";:SOUND 1,40:PRINT @ 0
,"SE PASA DEL LIMITE","LE ARREST
O!!!":SOUND 255,30
750 CLS0:PRINT @ 194,"LE RETIRO
EL CARNET";:PLAY "O3T5BAGFEDCO-B
AGFEDCO-BAGFEDC";:RUN
```

GRAFICOS DE ALTA RESOLUCION

Otra forma bastante sencilla de perder un juego es chocando frontalmente contra otro vehículo (= CIAN (color 6)). Es esta una buena ocasión para empezar a hablar de los gráficos de alta resolución. Lo primero que tenemos que considerar es cómo trazar un dibujo que represente al vehículo contrario (que puede ser un camión pesado,

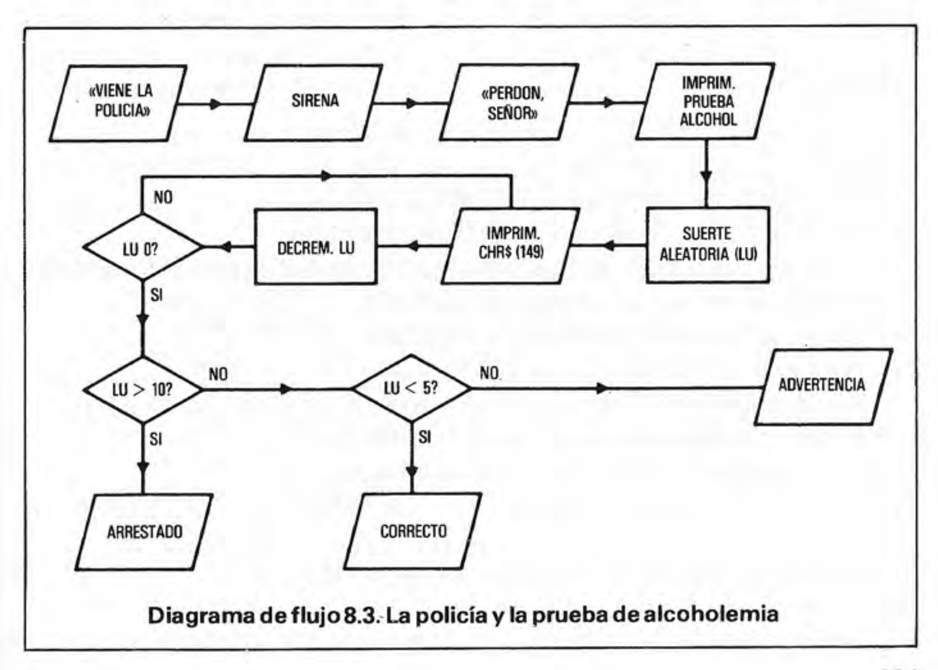


Diagrama de flujo 8.3. La policía y la prueba de alcoholemia

por ejemplo). Como se va acercar directa y frontalmente hacia nosotros, hemos de incrementar su tamaño progresiva y rápidamente. Utilizaremos para ello la instrucción DRAW ya que así la escala del dibujo puede incrementarse cada vez que se trace. Ahora debemos decidir qué modo PMODE emplear. Para este caso el detalle del dibujo no es esencial. Por tanto, podemos utilizar el modo de color de más baja resolución (PMODE1) que tiene la ventaja de ocupar tan sólo dos páginas gráficas en la memoria.

Debemos preparar el diseño del vehículo sobre una cuadrícula de alta resolución. Aunque la pantalla de alta resolución en el modo PMODE1 sólo tiene 128 × 96 elementos, sigue direccionándose como si tuviera 256 × 192, pero cada vez quedan afectados cuatro puntos de forma que si, por ejemplo, hacemos PSET(0,0,4) en PMODE1, en realidad, quedan afectados los puntos de coordenadas 0,0; 0,1; 1,0 y 1,1. Por consiguiente, siempre que dibujemos en modo PMODE1 resultará conveniente utilizar únicamente números pares ya que, de lo contrario, se producirían distorsiones cuando cambiáramos la escala del dibujo.

No resulta nada fácil reinterpretar las cadenas de las instrucciones DRAW. Para comprender un poco mejor cómo funcionan vamos a ver en detalle, paso a paso, cómo se dibuja la figura del camión. Empezando por la esquina superior izquierda, el camión que aparece en la figura 8.1 puede dibujarse como sigue:

1) Línea exterior principal «R28D40L28U40».
2) Parte superior de la cabina «D16R28D2L28»
   (Con D16 se dibuja sobre una línea ya existente pero esto requiere menos espacio que mover un blanco.)
3) Parte inferior del parabrisas «D10R28».
4) Faro derecho «BM-4, + 4;L4D4R4U4».
   (Es necesario realizar un movimiento sin dibujar nada para que el faro quede aislado del resto del dibujo.)
5) Faro izquierdo «BM−16, +0;L4D4R4U4».
6) Parrilla del radiador «BM + 2, + 0;R8BM + 0, + 2;L8BM + 0, + 2;R8».
7) Ruedas «BM + 8, + 4;D4L4U4L16D4L4U4».
8) Conductor.
   «BM + 0, − 12;U4R6D4U4L2U2L2D2L2D4R1U2R4D2L3U2».

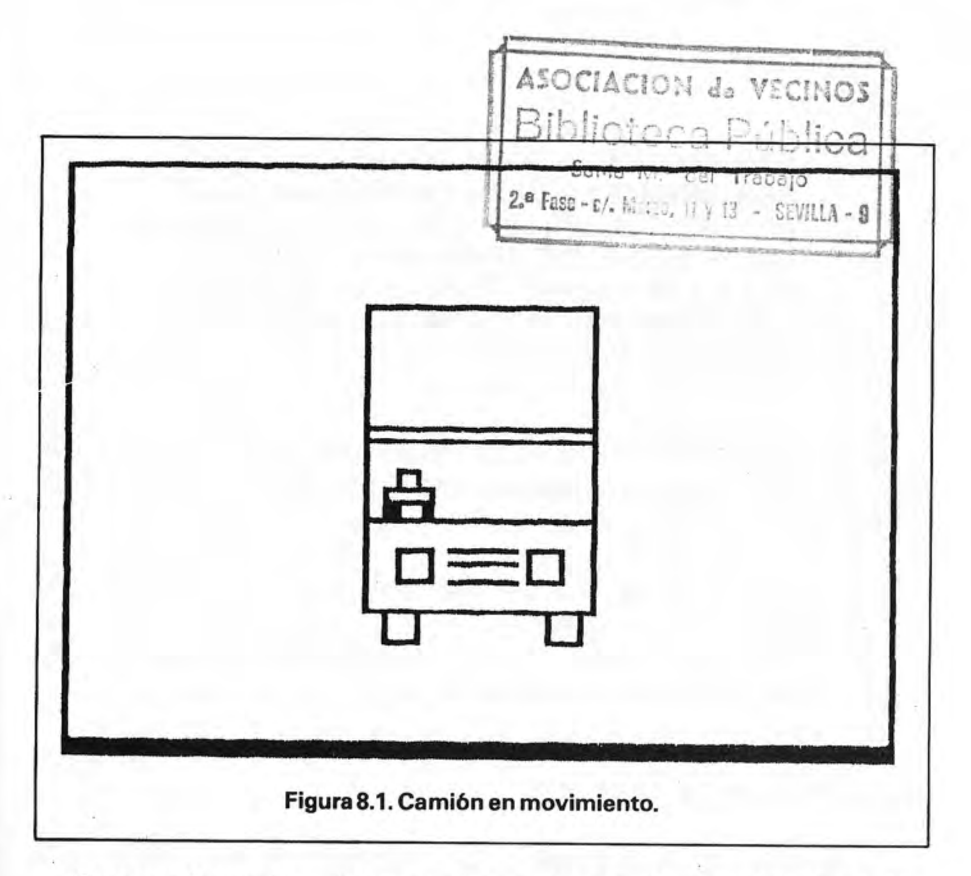


Figura 8.1. Camión en movimiento.

Compruebe todo esto y asegúrese de que lo ha encontrado correctamente. Añada una línea provisional (la 899) que mantenga el dibujo en la pantalla durante un cierto tiempo.

```
850 PMODE 3,1:SCREEN 1,0:PCLS
860 DRAW"R28D40L28U40D16R28D2L28
D10R28BM-4,+4;L4D4R4U4BM-16,+0;L
4D4R4U4BM+2,+0;R8BM+0,+2;L8BM+0,
+2:R8BM+8,+4;D4L4U4L16D4L4U4BM+0
,-12;U4R6D4U4L2U2L2D2L2D4R1U2R4D
2L3U2"
899 GOTO 899
```

EL AUMENTO DEL TAMAÑO DE LOS GRAFICOS

Esto nos dará un pequeño dibujo en el centro de la pantalla. Para aumentar su tamaño podemos emplear un bucle FOR-NEXT y añadir STR$(N) a la cadena de la instrucción DRAW.

```
855 FOR N=1 TO 20
860 DRAW"S"+STR$(N)+"R28D40L28U4
0D16R28D2L28D10R28BM-4,+4;L4D4R4
U4BM-16,+0;L4D4R4U4BM+2,+0;R8BM+
0,+2;L8BM+0,+2;R8BM+8,+4;D4L4U4L
16D4L4U4BM+0,-12;U4R6D4U4L2U2L2D
2L2D4R1U2R4D2L3U2"
875 NEXT N
```

Si ejecuta estas líneas se sorprenderá al ver aparecer a toda una caravana de camiones avanzando a través de la pantalla hacia usted. Se ha olvidado de dos cosas:

1) Hay que borrar la última figura dibujada antes de trazar la siguiente.
2) Tiene que empezar a trazar cada dibujo por la misma posición relativa en la pantalla.

EL BORRADO DE DIBUJOS

La posición de inicio del dibujo por omisión es la que corresponde al centro de la pantalla (128,96) pero cualquier nuevo dibujo debe empezar en la última posición de dibujo que, en este caso, es parte del conductor. La manera más fácil de asegurar que siempre empezamos a dibujar por el mismo punto inicial es colocar BM128,96 al principio de la definición de DRAW y, después, especificar el punto de inicio mediante un BM relativo desde ahí. Cualquier movimiento relativo de un espacio en blanco (especificado como + o —) se incrementará o disminuirá como el resto de la cadena en la que aparezca según la escala. Para mantenernos en el centro debemos hacer al principio BM—14,—20. La forma más sencilla de borrar el dibujo es poniendo PCLS inmediatamente antes de la instrucción DRAW.

```
855 FOR N=1 TO 20:PCLS
860 DRAW"S"+STR$(N)+"BM128,96;BM
-14,-20;R28D40L28U40D16R28D2L28D
10R28BM-4,+4;L4D4R4U4BM-16,+0;L4
D4R4U4BM+2,+0;R8BM+0,+2;L8BM+0,+
```

```
2;R8BM+8,+4;D4L4U4L16D4L4U4BM+0,
-12;U4R6D4U4L2U2L2D2L2D4R1U2R4D2
L3U2"
875 NEXT N
```

Con esto se obtiene una figura cada vez pero todavía quedan un par de pequeños problemas por resolver. Si detiene la ejecución del programa apretando SHIFT @ podrá darse cuenta de que algunas líneas no están totalmente en el lugar que debieran estar. Esto se arregla fácilmente haciendo que el bucle FOR NEXT vaya de 2 a 21 y avance de 2 en 2 (STEP 2). La distorsión que se puede apreciar cuando el dibujo se hace muy grande es inevitable pero subraya la idea de choque. La animación no. es muy impresionante dado que la instrucción PCLS es un instrumento de trabajo un poco primitivo. Por este motivo podemos apreciar aún cómo se dibuja la figura cada vez (aunque por breve tiempo).

Lo que tenemos que hacer es dibujar la figura en diferentes páginas gráficas (la 3 y la 4) y luego copiar estas páginas mediante PCOPY sobre las páginas 1 y 2. Sólo la línea 850 contiene un comando SCREEN (que sigue a PMODE 1,1) de forma que siempre estamos viendo la figura que está en las páginas 1 y 2. Aunque la página que se direcciona cambia por efecto de la línea 855, no veremos qué hay en ella hasta que la copiemos sobre las páginas 1 y 2 mediante PCOPY (en la línea 865). En la línea 875 PCLS sirve para borrar las páginas 3 y 4 antes de preparar el siguiente dibujo.

```
855 PMODE 3,5:FOR N=2 TO 21 STEP
 2
865 PCOPY 3 TO 1:PCOPY 4 TO 2
875 PCLS:NEXT N
```

LA SIMULACION DE LA ROTURA DE UN PARABRISAS

Esto hace que las cosas se vean mucho mejor. Vamos a añadir ahora las instrucciones necesarias para simular la rotura de nuestro parabrisas utilizando el comando CIRCLE. Obsérvese cómo, en vez de círculos, se dibujan elipses al haber fijado la razón entre la altura y la anchura en 0.5 (Fig. 8.2).

```
880 PMODE 1,1:FOR N=1 TO 200 STE
P 10:CIRCLE(100,140),N,2,0.5:NEX
T N
```

EFECTO DE DESVANECIMIENTO DE FIGURAS, DEL EXTERIOR AL INTERIOR DE LA PANTALLA

Como toque final, haremos que desaparezca la figura de la pantalla desvaneciéndose desde el exterior hacia el interior. Así simularemos la pérdida de conciencia que sufre el conductor (usted) (Fig. 8.3).

```
885 FOR N=60 TO 1 STEP-1:CIRCLE(
128,96),(N*3),3:NEXT N
```

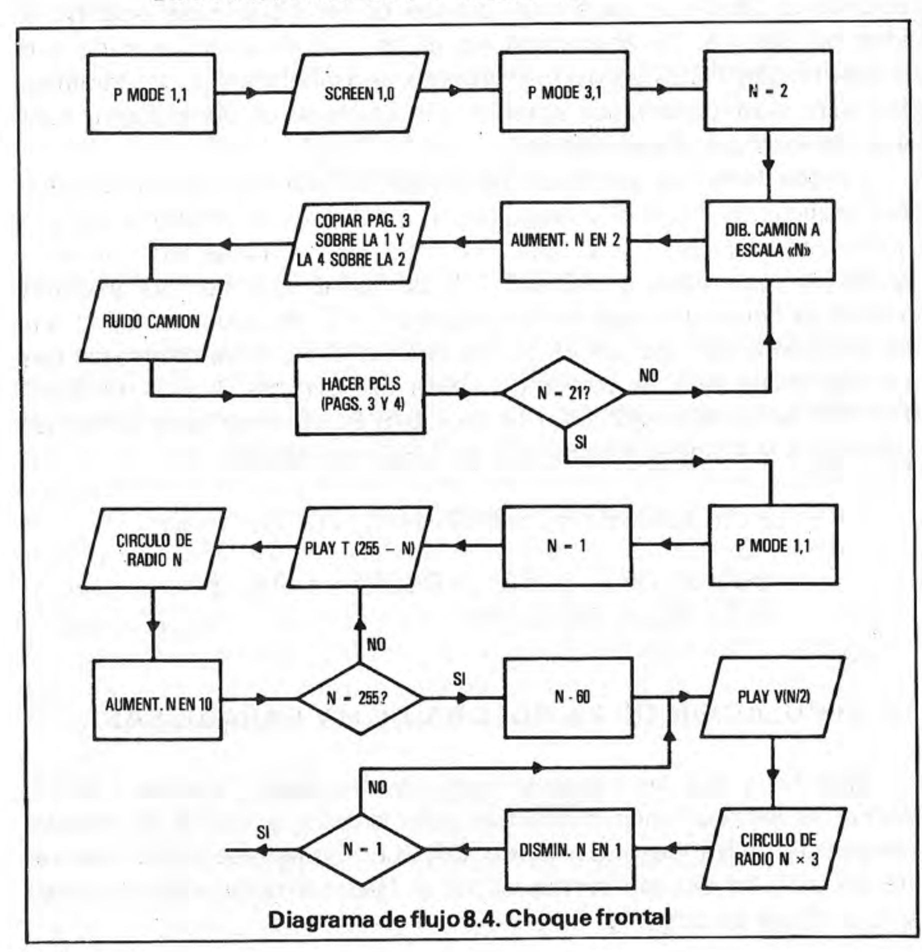


Diagrama de flujo 8.4. Choque frontal

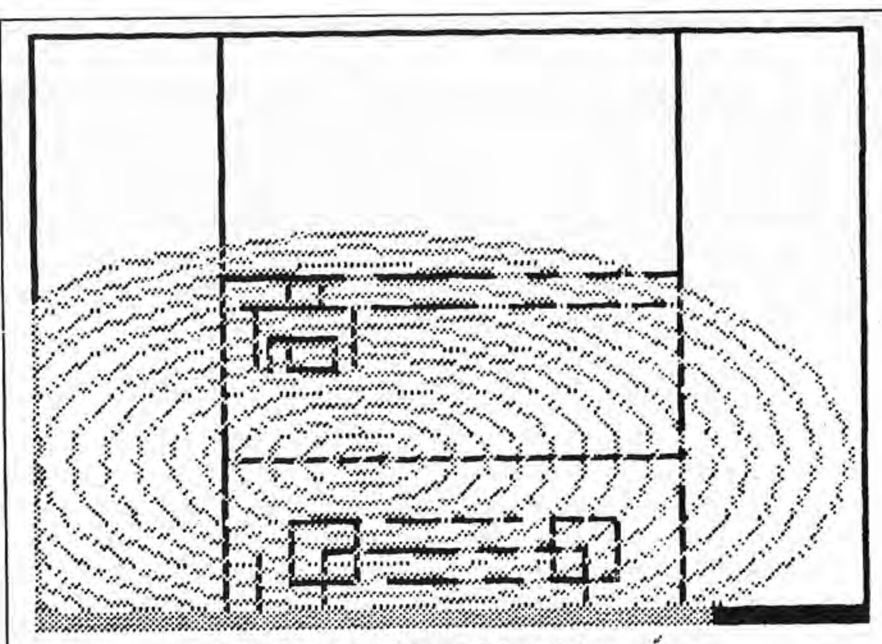


Figura 8.2. Rotura del parabrisas

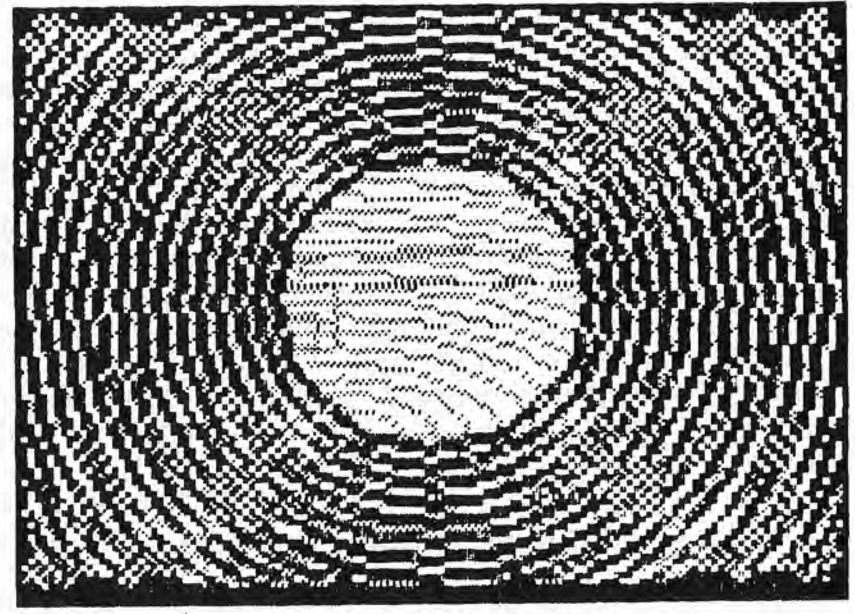


Figura 8.3. Pérdida de conciencia

INTEGRACIÓN DE GRAFICOS Y DE SONIDOS

Como refinamientos finales añadiremos tres sonidos diferentes. El primero se relaciona con la inminente colisión; el segundo se oye al romperse el parabrisas y el tercero corresponde a la palpitación que usted nota dentro de su cabeza al perder la conciencia. Todos estos sonidos se consiguen mediante instrucciones PLAY integradas con el aumento de tamaño del dibujo, de forma que cambian con cada movimiento. Cuando se integran gráficos y sonido es esencial comprobar que no se pueden generar valores incorrectos.

```
870 PLAY "V"+STR$((INT(N*1.5))+"
O1T255CDCD"
880 PMODE 1,1:FOR N=1 TO 255 STE
P10:PLAY "T"+STR$(255-N)+"O1CO5B
O3C":CIRCLE(100,140),N,2,0.5:NEX
T N
885 FOR N=60 TO 1 STEP-1:PLAY"V"
+STR$(INT(N/2))+"O1T255DCDCDCDC"
:CIRCLE(128,96),(N*3),3:NEXT N
```

El primer sonido (línea 870) es una breve audición «CDCD» que aumenta el volumen en 1.5*N veces con cada nuevo trazado del camión. El mayor valor de N es 21 y 21*1.5 = 31.5 lo que nos da un volumen incorrecto pero, como INT siempre aproxima por defecto, se obtiene 31. El ritmo del ruido asociado a la rotura del parabrisas se reduce a medida que pasa el tiempo. Por el contrario, el tamaño de los círculos aumenta. Esto se debe a que el ritmo se cambia mediante 255 — N mientras que los círculos aumentan en cada vuelta del bucle por efecto de N. El volumen final del sonido desciende al tiempo que los círculos disminuyen. N tiene que dividirse por dos para obtener valores correctos.

Quizás ya ha llegado el momento de apartarse de la carretera mortal y enfrentarse a los peligros que esperan al desprevenido jugador en el interior de un laberinto.

GRAFICOS ANIMADOS, EN BAJA RESOLUCION

Una sorpresa muy desagradable que podríamos añadir al programa de la «Ventana en el laberinto» sería la de incluir un duende encar-

gado de robarnos el dinero. Para ello podríamos emplear una forma sencilla de gráficos animados en baja resolución (Diagrama de flujo 8.5). Primero hay que diseñar una figura (Fig. 8.4). Hemos escogido la más sencilla posible. La parte principal de esta figura (A) es constante y se puede apreciar en todas las distintas posiciones que se ilustran (B—E). Para conseguir las distintas posiciones de las figuras se añaden las modificaciones necesarias. En baja resolución los puntos de los gráficos se especifican mediante las coordenadas X,Y y se activan mediante la instrucción PSET. Las características del cuerpo principal del duende se guardan en instrucciones DATA situadas en las líneas 5000 y 5100 en forma de coordenadas. Estas se leen por parejas mediante instrucciones READ X,Y y se emplean para dar a unos puntos determinados el color especificado por RND(8). El duende puede, por tanto, ser de cualquier color. Colóquese la línea provisional 5999 GOTO 5000 y hágase RUN 5000 para ver qué tal aparece la figura. Hay que asegurarse de que no se intentan leer más datos de los que se incluyen en las instrucciones DATA. El RESTORE que hay al final es imprescindible si se quiere volver a leer los mismos datos otra vez antes de emplear RUN ya que esta última instrucción vuelve a inicializar el puntero al principio de la información contenida en la instrucción DATA.

```
5000 DATA3,0,4,0,5,0,6,0,7,0,3,1
,5,1,7,1,3,2,4,2,5,2,6,2,7,2,3,3
,7,3,0,4,1,4,2,4,3,4,4,4,5,4,6,4
,7,4,8,4,9,4,10,4
5010 DATA3,5,4,5,5,5,6,5,7,5,3,6
,4,6,5,6,6,6;7,6,3,7,4,7,5,7,6,7
,7,7,3,8,4,8,6,8,7,8,3,9,4,9,6,9
,7,9,3,10,4,10,6,10,7,10,1,11,2,
11,3,11,4,11,6,11,7,11,8,11,9,11
5020 C=RND(9):CLS0:FOR N=1 TO 61
:READ X,Y:SET(X,Y,C):NEXT N:REST
ORE
```

LA ANIMACION DE GRAFICOS

El paso siguiente consiste en imprimir un mensaje que nos advierta de que hemos encontrado un duende. Luego él mismo nos pedirá

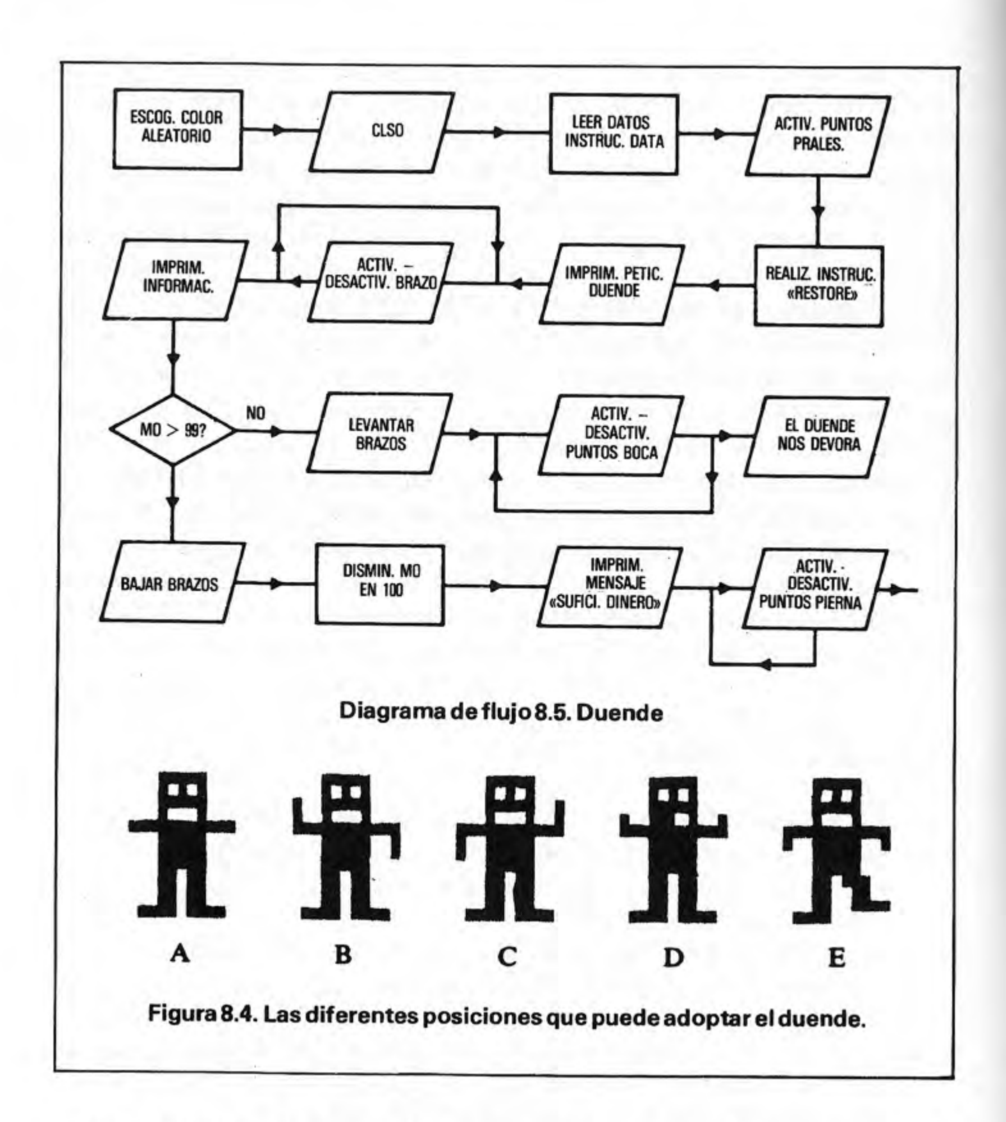


Diagrama de flujo 8.5. Duende

Figura 8.4. Las diferentes posiciones que puede adoptar el duende.

dinero. Hemos de poner las instrucciones necesarias para que el duende gruña y agite los brazos con notables muestras de impaciencia por cobrar. Las figuras B y C de la figura 8.4 muestran las dos distintas disposiciones de los brazos que son necesarias para crear el efecto de movimiento. Primero se levanta el brazo izquierdo (0,2 y 0,3) y se baja el derecho (10,5 y 10,6). Esta posición se mantiene durante el tiempo que se ejecuta la instrucción PLAY. El ritmo

de ésta depende del número de vueltas completas que se hayan ejecutado. Inmediatamente después se vuelve a dar su valor a los puntos (RESET) de forma que desaparezcan antes de dibujar el brazo izquierdo arriba y el brazo derecho abajo. Compruébese que la animación es mejor si los puntos anteriores no reciben otro valor hasta justo un momento antes de dar su valor a los nuevos. Finalmente, la parte inferior de la pantalla se ennegrece llenándola con CHR$(128) utilizando STRING$ antes de que se compruebe la cantidad de dinero de la que el jugador dispone ante la petición que le hace el duende.

```
5030 FOR N=30 TO 1 STEP -5:PRINT
 @ 224,"TE HAS ENCONTRADO CON UN
 DUENDE CON","UNA TREMENDA AFICI
ON AL DINERO":PRINT @ 320,"A MEN
OS QUE LE DES $100","TE DEVORARA
!!"
5040 SET(0,2,C):SET(0,3,C):SET(1
0,5,C):SET(10,6,C):PLAY"T"+STR$(
N)+"O1EC":RESET(0,2):RESET(0,3):
RESET(10,5):RESET(10,6):SET(0,5,
C):SET(0,6,C):SET(10,2,C):SET(10
,3,C):RESET(0,5):RESET(0,6):RESE
T(10,2):RESET(10,3):NEXT N
5050 PRINT @ 224,STRING$(160,128
):PRINT @ 224,"TIENES $";MO:IF M
O>99 THEN 5070
```

Como no hemos hecho nada para que el jugador disponga inicialmente de una cierta cantidad de dinero ni tampoco le hemos dado medio alguno de conseguirlo durante el desarrollo del juego, queda claro cuál será el resultado de esa última comprobación. Iremos a parar a la línea 5060 que hace que el duende levante los brazos en señal de victoria y mueva su lengua de izquierda a derecha, relamiéndose de gusto. Mientras tanto, suena una charanga triunfal que se convierte en seguida en el sonido de unas feroces mandíbulas que se abren y cierran. El bucle de retardo (Z) que hay en medio es imprescindible si se quiere ver cómo se mueve la lengua del duende.

```
5060 SET(0,2,C):SET(0,3,C):SET(1
0,2,C):SET(10,3,C):FOR N=1 TO 20
```

```
 STÉP 2:FOR M=4 TO 6:SET(M,3,C):
FOR Z=1 TO 100:NEXT Z:RESET(M,3)
:NEXT M:PRINT @ 384,"EL DUENDE T
E ACABA DE DEVORAR":SCREEN 0,1:P
LAY "T"+STR$(N*8)+"02DECDECDEC":
NEXT N:RUN
```

Si se realizan algunos cambios para poder disponer de dinero (una solución provisional podría consistir en entrar MO = 100 como instrucción directa y hacer luego GOTO 5000) se podrá llegar a la otra rutina de éxito. En ésta el duende baja los brazos, su dinero disminuye y el jugador recibe cierta información; luego, el duende demuestra su enfado dando patadas contra el suelo cada vez más fuertes con su pie derecho.

```
5070 SET(0,5,C):SET(0,6,C):SET(1
0,5,C):SET(10,6,C):MO=MO-100:PRI
NT @ 352,"TIENES SUFICIENTE PARA
 PAGAR AL DUENDE":PRINT @ 384 ,"
AHORA TE QUEDAN $";MO;
5080 FOR N=5 TO 25:PLAY"T"+STR$(
N)+"01C":RESET(6,9):RESET(6,10):
RESET(6,11):RESET(7,11):RESET(8,
11):RESET(9,11):SET(8,8,C):SET(8
,9,C):SET(8,10,C):SET(9,10,C):SE
T(10,10,C)
5090 SET(6,9,C):SET(6,10,C):SET(
6,11,C):SET(7,11,C):SET(8,11,C):
SET(9,11,C):RESET(8,8):RESET(8,9
):RESET(8,10):RESET(9,10):RESET(
10,10):NEXT N
```

Para emplear esta rutina en su programa sólo tiene que realizar los cambios necesarios para poder contar con el dinero y, además, tiene que añadir una instrucción RETURN al final de la línea 5090.

EL DIBUJO DE OBJETOS A PARTIR DE UN PUNTO CENTRAL

Otro ejemplo del empleo de gráficos de baja resolución muy sencillos es la siguiente rutina que dibuja una terrible araña (Diagrama de

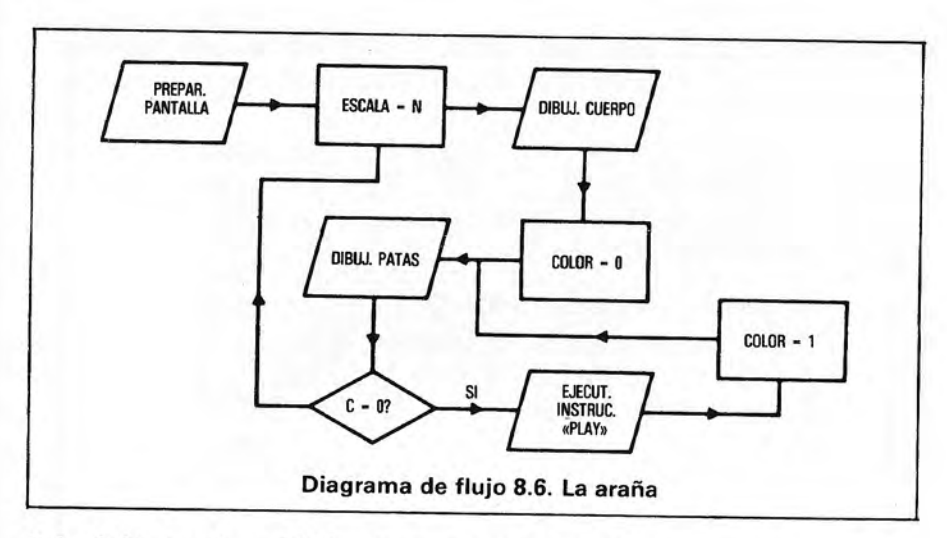


Diagrama de flujo 8.6. La araña

flujo 8.6). La línea 7010 dibuja el cuerpo de la araña y sus antenas haciendo que su tamaño aumente progresivamente (Fig. 8.5). En este caso no borramos el dibujo del cuerpo antes de pasar a dibujar el

Figura 8.5. El cuerpo de la araña

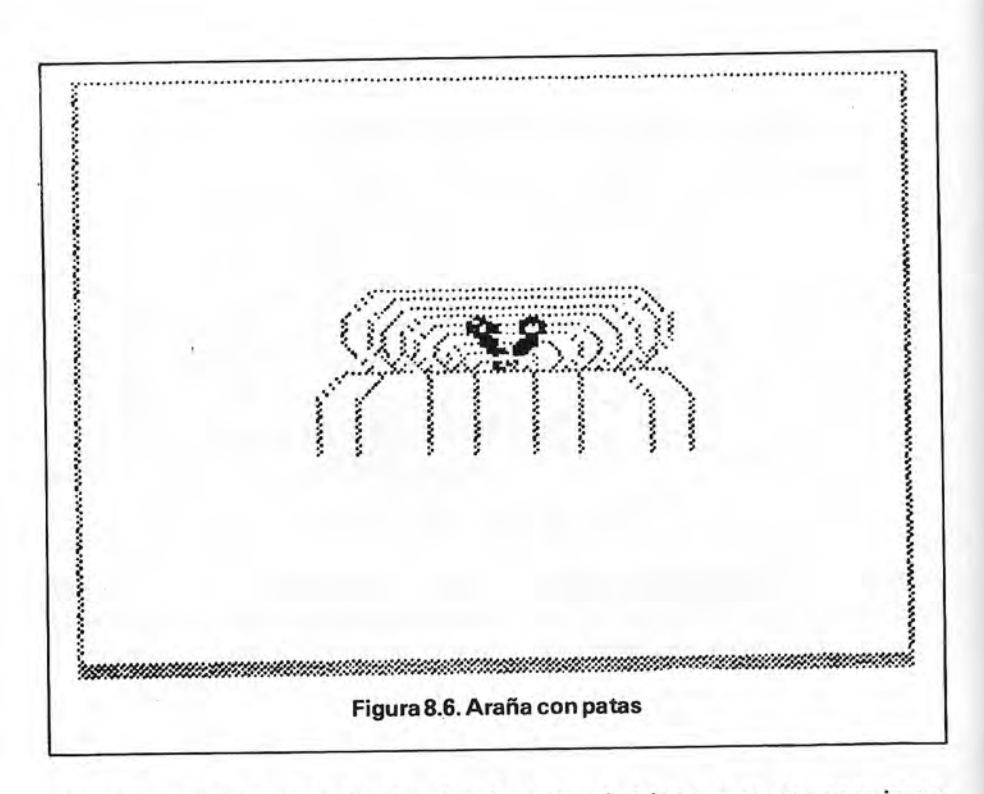


Figura 8.6. Araña con patas

siguiente ya que la serie de círculos concéntricos que conseguimos ahora le da una apariencia más sólida a la araña. La línea 7020 dibuja las patas. En esta línea hay un par de trucos interesantes (Fig. 8.6). Si se observa atentamente la cadena empleada por la instrucción DRAW se podrá entender el uso de N. Cuando N se coloca delante de un movimiento, la posición de dibujo a la que aplicamos la instrucción DRAW no se actualiza mientras dura ese movimiento. De esta forma, el movimiento siguiente empezará en el mismo lugar. Esta es una técnica muy útil para dibujar objetos que se originan a partir de un punto central ya que nos ahorra calcular cómo llegar a la siguiente posición. Las patas se dibujan dos veces en cada aumento de tamaño: una vez con color 0 y la otra con color 1. En teoría, el color 0 no estaría permitido en este modo de trabajo pero el Dragón cree que el 0, en este caso, es igual al 4 y, así, las patas se dibujan en color 4 (rojo) y luego en color 1 (verde). Como el fondo también es verde, esto significa, en la realidad, que las patas se «desdibujan» y que desaparecen. Cuando C = 0 se produce un so-

nido que atrae la atención sobre las patas rojas. Uno no se da cuenta de que algunas veces desaparecen totalmente.

```
7000 PCLS:PMODE 1,1:SCREEN 1,0
7010 FOR N=1 TO 16:DRAW"C4"+"S"+
STR$(N)+"L18H4U4E4R36F4D4G4L18C2
BM-4,-5LHUERFDGLBM+8,+0LHUERFDGL
BM-3,+5"
7020 FOR C=0 TO 1:DRAW"C"+STR$(C
)+"R4ND12R6ND12R6F4D8U8H4R6F4D8U
8H4L26ND12L6ND12L6G4D8U8E4L6G4D8
U8E4R22":IF C=0 THEN SOUND (N*15
),N
7030 NEXT C,N
```

LA ENTRADA DE CARACTERES DESDE EL TECLADO

La situación que sigue a la aparición de la araña depende del jugador y de la suerte. Si durante el recorrido por el laberinto se encuentra con el dragón que debe andar por ahí puede hacerlo aparecer y traérselo de vuelta a casa. La aparición del dragón se comprueba mediante la línea 200 que lleva a la rutina que se encarga de todo lo relacionado con el dragón (línea 4000). Nuestro dragón (Figura 8.7) está hecho de caracteres de distintos colores pero, como el sistema del Dragón 32 no le permite entrar caracteres directamente a través del teclado (a menos que escriba CHR$(N) o STRING$), es muy probable que el listado siguiente le produzca un cierto cansancio en ojos y dedos. De todas formas, le aseguramos que el resultado final vale la pena.

Cuando haya entrado todas las líneas que van hasta la línea 4080 haga RUN para comprobar cuál ha sido su precisión en la entrada de caracteres. Debería poder ver un dragón de aspecto algo enfermizo con una tira verde. Cuando haya añadido el resto de líneas comprobará que el color se anima bastante y que el dragón lanza fuego. Eche una mirada detenida al orden de los elementos de la cadena empleada por la instrucción PLAY y sabrá por qué, al principio, oye esos dos sonidos tan distintos. Cuando el dragón está en el laberinto con el jugador, el carácter U mayúscula (carácter (85), el que repre-

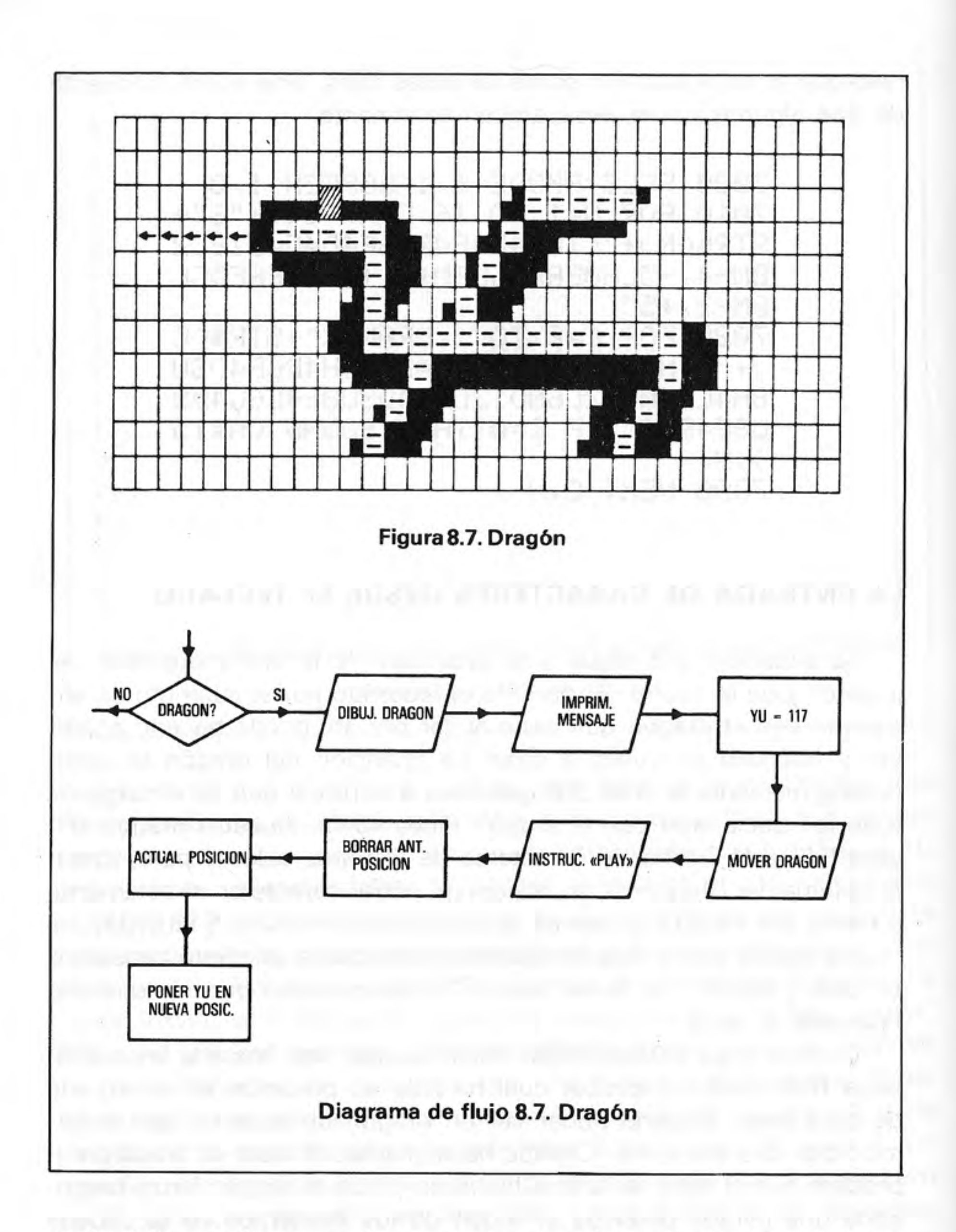


Figura 8.7. Dragón

Diagrama de flujo 8.7. Dragón

senta al jugador), se convierte en una «u» minúscula (en inversa) (carácter 117). Desde luego, hay que borrar de la posición anterior al dragón en cuanto usted se mueva (Diagrama de flujo 8.7).

```
200 IF B(V)=100 THEN 4000
4000 CLS0:PRINT @ 41,CHR$(241);C
HR$(243);CHR$(243);CHR$(159);CHR
$(243);CHR$(243);CHR$(242);STRIN
G$(4,128);CHR$(247);STRING$(5,61
);CHR$(249);
4010 PRINT @ 73,CHR$(255);STRING
$(5,61);CHR$(255);CHR$(242);STRI
NG$(2,128);CHR$(247);CHR$(61);CH
R$(255);CHR$(254);STRING$(2,252)
;CHR$(248);
4020 PRINT @ 106,CHR$(253);STRIN
G$(3,255);CHR$(61);CHR$(255);CHR
$(250);CHR$(128);CHR$(245);CHR$(
61);CHR$(255);CHR$(254);
4030 PRINT @ 141,CHR$(247);CHR$(
61);CHR$(254);CHR$(128);CHR$(241
);CHR$(61);CHR$(255);CHR$(254);
4040 PRINT @ 172,CHR$(245);CHR$(
255);STRING$(14,61);CHR$(255);CH
R$(242);
4050 PRINT @ 205,CHR$(253);STRIN
G$(2,255);CHR$(61);STRING$(10,25
5);CHR$(61);CHR$(255);CHR$(250);
4060 PRINT @ 238,CHR$(241);CHR$(
61);CHR$(255);CHR$(254);STRING$(
6,128);CHR$(247);CHR$(255);CHR$(
61);CHR$(255);CHR$(248);
4070 PRINT @ 269,CHR$(241);CHR$(
61);CHR$(255);CHR$(254);STRING$(
6,128);CHR$(241);CHR$(255);CHR$(
61);CHR$(255);CHR$(254);
4080 PRINT @ 384,"FELICIDADES ";
:PRINT @ 448,"HAS ENCONTRADO EL
DRAGON";:YU=117
4090 FOR M=1 TO 20:FOR N=1 TO 9:
PRINT @ (73-N),CHR$(127);:SCREEN
 0,1:PLAY STR$(N)+"V31T25505":NE
XT N:FOR N=1 TO 9:PRINT @ (64+N)
,CHR$(128);:SCREEN 0,1:NEXT N,M:
B(U)=143:U=V:B(U)=YU:GOTO 100
```

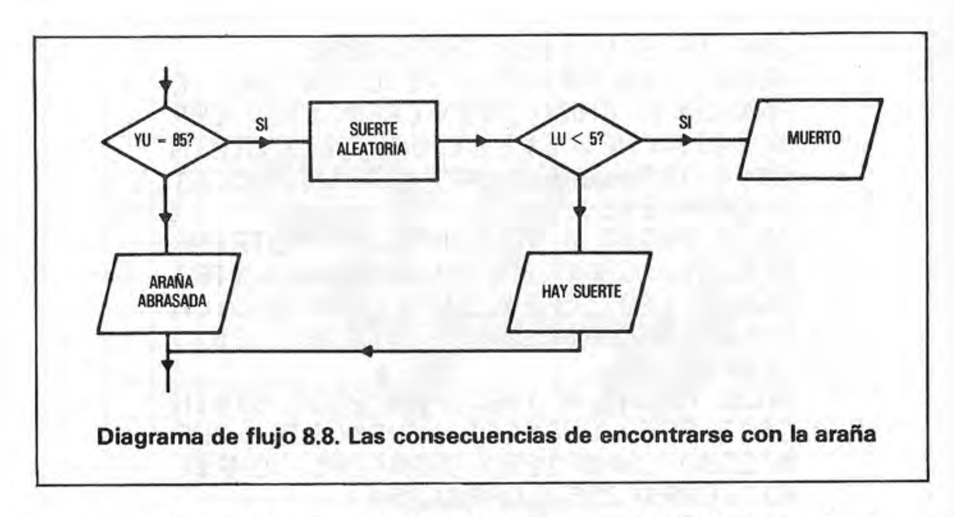


Diagrama de flujo 8.8. Las consecuencias de encontrarse con la araña

Si el jugador ha encontrado al dragón y se halla junto a él, tiene una gran ventaja ya que el dragón podrá abrasar a la araña con su tremendo aliento (Diagrama de flujo 8.8). El jugador podrá regresar seguramente y sin problemas al resto del programa. Incluso en el caso de que el dragón no esté con el jugador para echarle una mano, sus posibilidades de supervivencia rondan el 50 % si la suerte juega a su favor y la araña le deja a un lado.

```
7040 IF YU=85 THEN 7050 ELSE CLS
4:PRINT @ 160,"HAS TENIDO SUERTE
!":PRINT @ 224,"EL DRAGON HA ABR
ASADO A LA ARA/A!":FOR N=1 TO 30
00:NEXT N:GOTO 100
7050 LU=RND(10):IF LU<5 THEN 707
0
7060 CLS2:PRINT @ 128,"HAS TENID
O SUERTE!":PRINT @ 192,"A LA ARA
/A NO LE GUSTA TU SABOR":FOR N=1
0 TO 250 PRINT STEP 2:SOUND N,1:
NEXT N:GOTO 100
7070 CLS0:PRINT @ 128,"QUE TAL S
E ESTA EN LA PANZA DE UNA ARA/A
GIGANTE?":PRINT @ 256,"MALA SUER
TE!":FOR N=10 TO 200 STEP 10:SOU
ND N,2:NEXT N:RUN
```

UN MISMO OBJETO PARA DISTINTAS FUNCIONES

Es natural, donde hay buenas noticias también las hay malas. No se crea que tener un amigo dragón es siempre una buena idea: San Jorge se mueve, laberinto arriba y laberinto abajo, y está dispuesto no sólo a acabar con los dragones sino también con sus simpatizantes (Diagrama de flujo 8.9). Este es un buen ejemplo que ilustra cómo se puede emplear un mismo objeto para obtener resultados distintos. Si cuando llega San Jorge, el dragón no está con el jugador, tan sólo recibirá una severa advertencia.

```
8000 CLS4:PRINT STRING$(32,128):
PRINT"SOY SAN JORGE Y MATARE","A
 CUALQUIERA QUE AYUDE A ESE DRAG
ON!!":PRINT STRING$(32,185);:PRI
NT @ 320,"DEBES AVISARME SI LO V
ES":PLAY"V1DV+EV+FV+GV+DV+E":IF
YU<>85 THEN 100
8010 CLS0:PRINT @ 288,"MUERAN TO
DOS LOS AMIGOS DE LOS DRAGONES!!
",,"SAN JORGE!!!":PLAY"O1V20T2L4
GGL8GGL4B-AAGGF+G":RUN
```

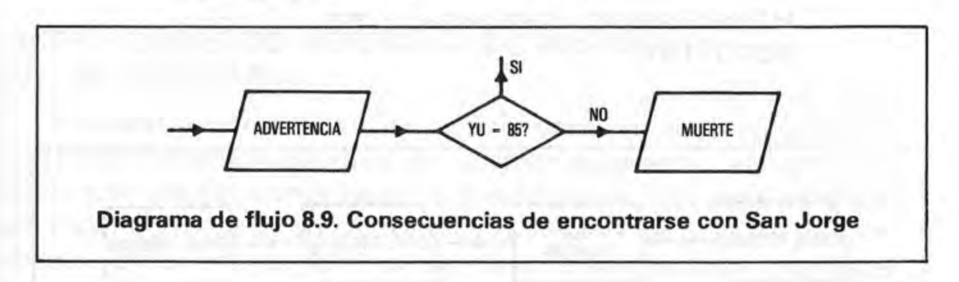


Diagrama de flujo 8.9. Consecuencias de encontrarse con San Jorge

ELIPSES CONCENTRICAS QUE ABSORBEN FIGURAS

Una alternativa más aceptable que la de morir, es la pérdida de alguna de sus pertenencias. ¿Qué tal si introducimos un remolino que engulla algo de su dinero o de su comida? (Figura 8.8) (Diagrama de flujo 8.10). Primero dibujamos la parte superior del cuerpo y de los brazos de una persona y luego añadimos un pequeño círculo que

represente la cabeza. Aparecen después una serie de elipses concéntricas que se tragan a la figura. El tamaño del remolino (número de círculos) depende de su suerte (LU) y este mismo factor determina cuántas pertenencias pierde. La pantalla de texto se borra tomando uno de los nueve colores posibles (del 0 al 8). Se consigue esto dividiendo LU por 12. La última instrucción de sonido ha de tener un + 1 para evitar la posibilidad de que se produzca un error FC cuando su suerte sea muy adversa.

```
6000 PMODE 3,1:SCREEN 1,0:PCLS:D
RAW"BM-10,+0U10H10E5F10R5E10F5G1
0D10":CIRCLE(125,75),8,4:LU=RND(
100):FOR N=1 TO (LU+0) STEP 3:CI
RCLE(129,96),N,RND(4),0.2:SOUND
N,1
6010 CLS INT(LU/12):PRINT @ 128,
"HAS ESCAPADO AL","REMOLINO PERO
HAS PERDIDO:",,INT(MO*LU/100);"
DINERO",,INT(FO*LU/100);"COMIDA"
6020 MO=MO-INT(MO*LU/100):FO=FO-
INT(FO*LU/100):PRINT @ 320,"TODA
VIA TIENES:",MO;"DINERO",,FO;"CO
MIDA":SOUND INT(MO*LU/50)+1,100:
GOTO 100
```

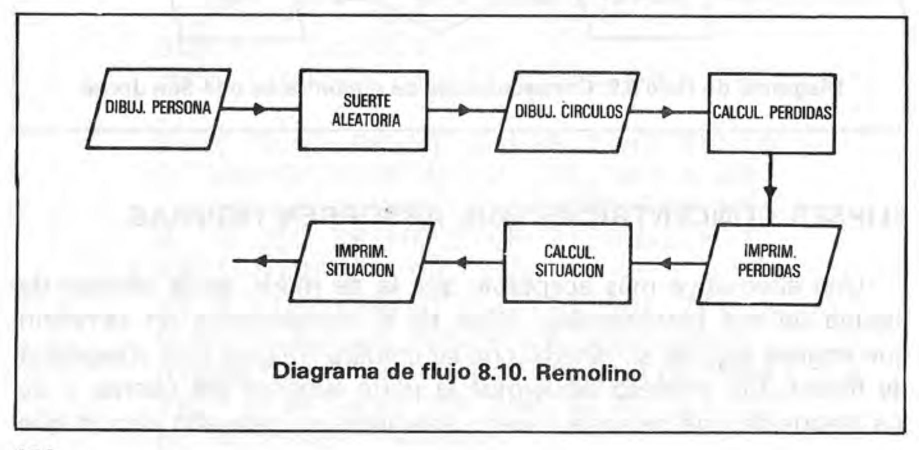


Diagrama de flujo 8.10. Remolino

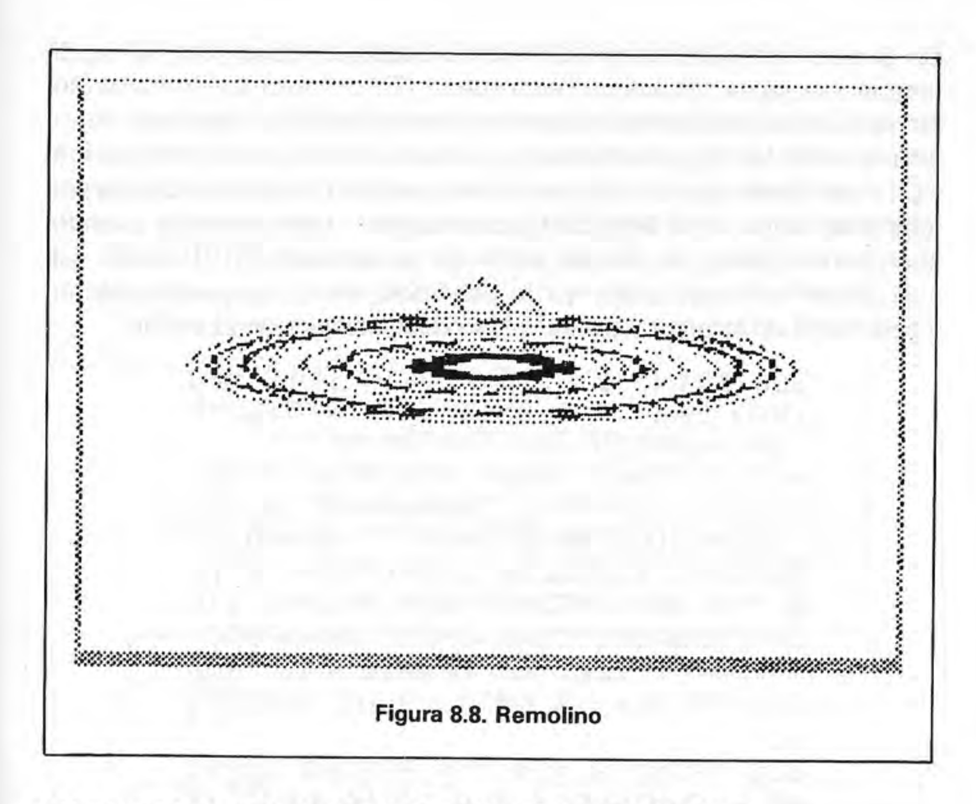


Figura 8.8. Remolino

DESPLAZAMIENTO REPENTINO DE POSICIONES EN UN PROGRAMA

El desplazamiento súbito de nuestra posición es otra idea interesante que podemos introducir en nuestros programas. Vamos a empezar por estudiar cómo incluir la posibilidad de que haya patinazos en nuestro programa automovilístico (Diagrama de flujo 8.11). Realmente patinar es muy fácil ya que todo lo que tenemos que hacer es desplazar nuestra posición hacia la derecha o la izquierda una distancia aleatoria. Se tiene en cuenta la velocidad que se lleva de forma que cuanto más rápido nos movamos más patinaremos. La velocidad se controla con la variable M. Ahora bien, estos patinazos tienen que afectar a las ruedas de alguna forma ¿no? Añadiremos una variable que registre el desgaste de sus neumáticos (TW). Se incrementará con cada patinazo según sea la velocidad que se lleve. Cuando el desgaste sea excesivo (> 49) se recibirá un mensaje que le comunicará que su neumático ha reventado. Tras un breve espacio

de tiempo se podrá continuar con la rueda de recambio (TY). Si también se gasta totalmente esta rueda (TY >) entonces empezarán los verdaderos problemas. Obsérvese que el mensaje final está intercalado entre los fragmentos de la cadena utilizada por la instrucción PLAY de forma que tan sólo se ve un pedacito cada vez. Si colocamos esta rutina en la línea 600, accederemos a ella tan sólo cuando choquemos contra un bloque verde en la carretera (CHR$(143)). La vida puede ser muy cruel, ya lo sabemos: ahora se puede patinar y salir fuera de la pista y produciendo desperfectos en el coche.

```
600 PRINT @ 0,"PATINAZO";:IF M=0
THEN E=E+(RND(3)-2):TW=TW+5:ELSE
 E=E+(RND(5)-3):TW=TW+10
610 IF TW<50 THEN 230 ELSE TW=0:
TY=TY+1:IF TY<2 THEN PRINT @ 32,
"NEUMATICO REVENTADO!!":SOUND 1,
50:GOTO 230:ELSE CLS4:PRINT @ 16
3,"TE HAS QUEDADO SIN RUEDAS";:P
LAY"O5BAGFEDCO4BAGFEDCO3BAGFEDC"
:PRINT @ 225,"EL GARAJE MAS PROX
IMO ESTA A 10 KMS";:PLAY"O2BAGFE
DC"
620 PRINT @ 424,"YA PUEDES EMPEZ
AR A CAMINAR";:PLAY"O1BAGFEDC":R
UN
```

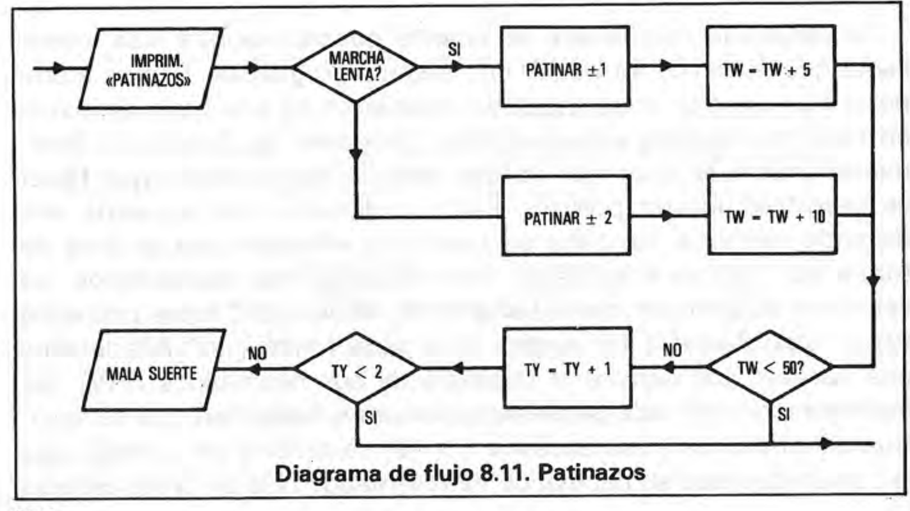


Diagrama de flujo 8.11. Patinazos

FORMA DE LOGRAR CONTRACCIONES ESPACIALES

El equivalente a un patinazo, en el programa de «la ventana en el laberinto», será una contracción del espacio que nos lleve a otro punto del laberinto (Diagrama de flujo 8.12). Se recordará que dispusimos unas cuantas trampas negras (CHR$(144)) en las paredes del laberinto. Estas trampas nos llevarán a la línea 3000 en donde entraremos en una contracción del espacio. Como nadie ha sobrevivido a una verdadera contracción espacial, sólo le podemos dar una impresión artística de lo que sucede.

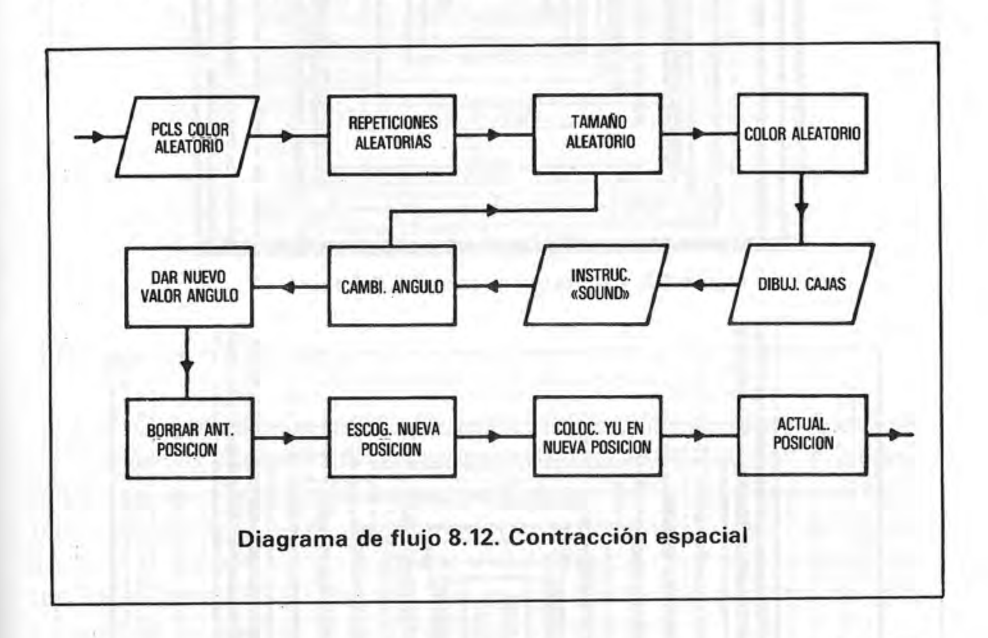


Diagrama de flujo 8.12. Contracción espacial

La pantalla toma un color aleatorio y entonces el jugador tiene la impresión de que se desplaza rápidamente por un corredor multicolor (Figura 8.9) al mismo tiempo que oye un extraño sonido. Esta secuencia puede repetirse pero con las paredes desplazadas en ángulo recto (parámetro correspondiente al ángulo (A)). Al escapar de la contracción espacial su posición anterior toma el color verde y el jugador es transportado a una posición aleatoria dentro del laberinto.

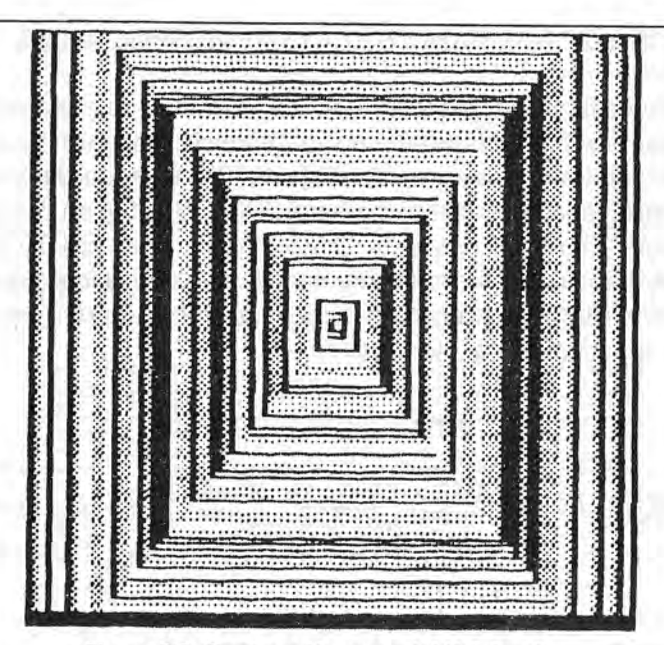


Figura 8.9. Contracción espacial (1.ª fase)

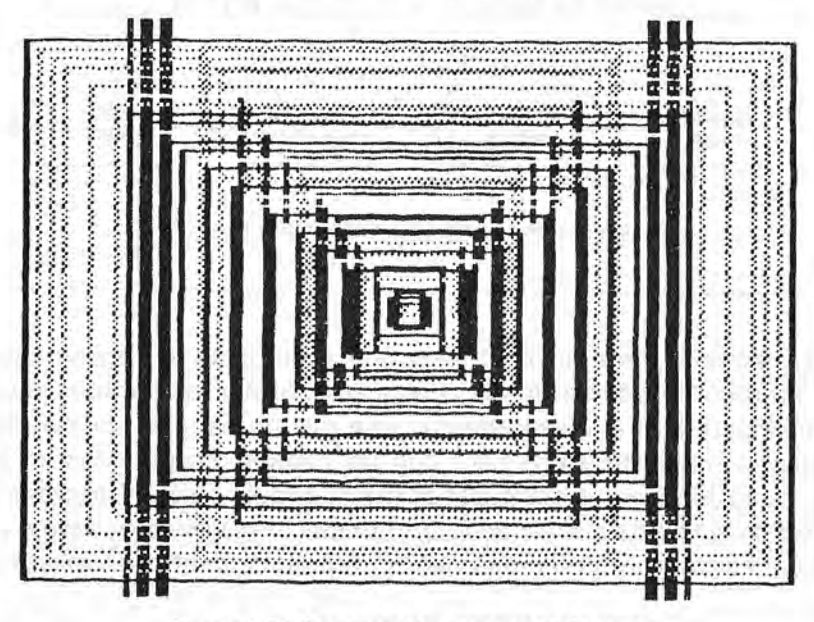


Figura 8.10. Contracción espacial (2.ª fase)

Justo un momento antes de recobrar la visión parcial que le ofrecía la ventana, el jugador podrá tener una visión total del laberinto. Así podrá saber más o menos dónde está. Obsérvese que el parámetro del ángulo vuelve a tener el valor 0 al final de la línea 3000 para que el dibujo siguiente no aparezca en la dirección equivocada.

```
3000 PCLS(RND(4)):PMODE 3,1:SCRE
EN 1,0:FOR NM=1 TO (RND(4)):FOR
N=1 TO 62 STEP NM:DRAW"S"+STR$(N
)+"C"+STR$(RND(3))+"BM-6,+8U16R1
2D16L12BM128,96":SOUND (NM*N),1:
NEXT N:DRAW"A"+STR$(RND(2)):NEXT
 NM:DRAW"A0"
3010 B(U)=143:U=RND(431)+128:B(U
)=YU:V=U
3020 CLS0:FOR N=97 TO 576:PRINT
CHR$(B(N));:NEXT N:FOR N=1 TO 50
0:NEXT N:CLS:GOTO 100
```

AHORA ES SU TURNO

Las consecuencias que sus actos tengan durante el desarrollo de los juegos pueden ser muchas y muy variadas. Hemos intentado mostrarle las diferentes maneras en que pueden presentarse en pantalla. Recuerde que el papel principal lo desempeña la imaginación junto al buen uso de los gráficos y del sonido. Aquí van unas cuantas ideas para hacer trabajar su cerebro. Estamos seguros de que a usted se le pueden ocurrir muchas otras más.

1) Prepare unas cuantas rutinas más que gestionen las consecuencias de chocar contra bloques de otros colores en «A toda marcha».
2) Vuelva a diseñar la rutina de colisión para que el camión se aparte de su camino en el último momento.
3) Utilice los gráficos de baja resolución para crear la figura animada de un monstruo de dos cabezas y seis brazos que se moverá siguiendo una secuencia aleatoria.

4) Vuelva a diseñar el remolino para que se convierta ahora en arenas movedizas en las cuales se hunde lentamente una figura humana.
5) Intente realizar algunas secuencias animadas en modo PMODE 1 recordando que PSET y PRESET son los equivalentes en alta resolución de SET y RESET y que puede volver a utilizar los datos que colocó en la instrucción DATA del programa del duende.

Capítulo IX

LOS JUEGOS ESPACIALES

Aunque no se puede conseguir con el BASIC la misma rapidez de los juegos electrónicos escritos en lenguaje máquina, es posible tener un poco de espectáculo extraterrestre recurriendo a las instrucciones gráficas GET y PUT que nos permiten mover, de forma continua y sin interrupciones, figuras en la pantalla, en alta resolución. Pero para utilizar estas instrucciones necesitamos disponer de algo que mover. Empezaremos, por tanto, dibujando un par de naves espaciales no muy complicadas.

```
10 PMODE 3,1:SCREEN 1,1:PCLS
100 DRAW"C8BM8,8D5R5U5L2U4D4L2D5
L2D2R8U2L2"
110 CIRCLE(40,10),10,6,0.5:PAINT
(40,10),7,6:PSET(40,10,5)
1000 GOTO 1000
```

MENOS MEMORIA Y MAS RAPIDEZ EN LA EJECUCION DE DE UN PROGRAMA

Ahora necesitamos definir matrices lo suficientemente grandes como para almacenar todos los detalles gráficos de esas dos figuras. Así podremos utilizar la instrucción GET. Resulta siempre conveniente aplicar la instrucción GET sobre una mínima superficie posible de la pantalla. Así, no sólo conseguimos ahorrar memoria sino que también aumenta la rapidez de la ejecución y se hace posible desplazar con facilidad objetos muy próximos entre sí. El área más pequeña que podemos asignar a una nave espacial comprenderá los puntos 4

a 14 del eje X y los puntos 4 a 16 del eje Y. Las cifras correspondientes a la segunda nave espacial son del 31 al 52 y del 5 al 20.

EL TOTAL DE BYTES PARA REPRESENTAR UNA FIGURA EN DETALLE

El manual dice que el tamaño de la matriz correspondiente a las figuras se debe determinar tomando la anchura sobre el eje X y la altura sobre el Y (contadas ambas en puntos de la pantalla) y multiplicando ambas cantidades. En realidad, esto no es totalmente cierto ya que cada elemento de la matriz consta de cinco bytes y en un solo byte hay información referente a más de un punto de la pantalla.

El tamaño mínimo real depende principalmente del modo PMODE en que se trabaje y de la forma del área afectada por la instrucción GET. Para áreas cuadradas, se puede calcular como sigue (haciendo el redondeo de las cantidades por aproximación al número mayor inmediato): se calcula el número de puntos de la pantalla que se requieren. Si se trabaja en PMODE 0 hay que dividir por 80 la cifra conseguida. Si se trabaja en PMODE 1 ó 2, se divide por 80 y en PMODE 3 ó 4 se ha de dividir por 40. Así, se obtiene el número total de bytes necesarios para representar la figura en detalle. Basta con dimensionar la matriz con un 0 como primera cifra y, como segunda, el número que acabamos de calcular.

Si la figura que deseamos representar no es cuadrada hay que tratarla como si fuera un cuadrado de la mayor dimensión. (Si el programa no funciona correctamente, con los números que se han obtenido de esta forma, se comprobará que se resuelve el problema añadiendo un nuevo elemento a la matriz.)

El ahorro de memoria que estos cálculos aportan puede comprobarse comparando el tamaño que, según el manual, debería tener la primera matriz (A2) (120 = 10 × 20 puntos) con el que tiene en realidad (3). En cuanto a la matriz utilizada para representar la segunda figura, que es un poco extraña puesto que es bastante ancha con relación a su altura, sólo son necesarios nueve elementos (A3). También definimos una matriz vacía (A1) con el mismo tamaño que la mayor de las dos figuras. Lo emplearemos para borrar cosas cuando sea necesario. No es necesario introducir en A1 ninguna infor-

mación con la instrucción GET si se está utilizando como fondo el color de número más bajo puesto que A1 ya está llena de ceros. Una vez almacenadas las figuras en las correspondientes matrices, podemos eliminarlas de la pantalla haciendo PCLS. Ahora podemos empezar a construir el juego propiamente dicho.

```
20 DIM A1(0,9):DIM A2(0,3):DIM A
3(0,9)
200 GET(4,4)-(14,16),A2,G
210 GET(31,5)-(52,20),A3,G
220 PCLS
```

LA UTILIZACION DEL MANDO PARA JUEGOS (JOYSTICKS)

Para recrear la imagen de la figura sobre cualquier punto de la pantalla necesitamos utilizar la instrucción PUT especificando las coordenadas del punto en que deseamos que aparezca. También tenemos que decidir qué hacer con cualquier otra figura que se encontrara ya en esas coordenadas. La mejor forma de moverse por el espacio es mediante un mando para juegos: las rutinas que describimos de aquí en adelante lo emplean. Si no se dispone de un mando para juegos consúltese en capítulos anteriores las modificaciones que deben hacerse para sustituir las comprobaciones referidas a los mandos de juego por otras que comprueben las teclas que se pulsan. Leeremos los valores de JOYSTK(0) y JOYSTK (1) y se los daremos a las variables J0 y J1 que emplearemos como coordenadas para la instrucción PUT. Los límites de movimiento de los mandos para juego son 0 y 63. La pantalla, en cambio, está «mapeada» del 0 al 255 y del 0 al 191. Por consiguiente, realizaremos la conversión necesaria multiplicando J0 por 4 y J1 por 3. Lo más sencillo que debemos hacer es PSET en toda la pantalla.

```
300 J0=JOYSTK(0)*4:J1=JOYSTK(1)*
3
380 PUT(J0,J1)-(J0+10,J1+12),A2,
PSET
1000 GOTO 300
```

Si se hace RUN se comprobará que la figura se mueve en respuesta a los movimientos que se imprimen a la palanca del mando para juegos. Sin embargo, se comprobará también que la figura deja tras de sí una estela ya que hemos olvidado borrar las anteriores figuras antes de escribir las siguientes. Ninguno de los parámetros de acción alternativos nos será de utilidad aquí. Si no lo cree, intente modificar la línea 380 y verá qué pasa. Lo que necesitamos hacer es aplicar a la anterior posición de la figura la instrucción PUT con la matriz que habíamos dejado vacía. Lo más sencillo es colocarla mediante PUT en las mismas coordenadas, haciendo PSET inmediatamente después de nuestra figura como se ve en la línea 450.

```
450 PUT(J0,J1)-(J0+10,J1+12),A1,
PSET
```

COMPARACION DE VARIABLES PARA DETECTAR MOVIMIENTOS

Se comprobará que, ahora, la figura se mueve pero que destellea a gran velocidad lo cual hace difícil verla. ¿Qué le parece si la dejamos en la pantalla a menos que nos queramos mover? Para hacer esto necesitamos comprobar si el mando para juegos joystick se ha movido o no. Podemos emplear un par de variables T0 y T1 para guardar o almacenar las anteriores posiciones de la figura. Compararemos el valor de estas variables con las nuevas y así sabremos si se ha realizado un movimiento o no (Diagrama de flujo 9.1). Hemos de leer T0 y T1 al principio, antes que J0 y J1. Tenemos que hacerlo en el primer ciclo si no queremos que la figura aparezca en la posición 0.0.

```
230 T0=JOYSTK(0)*4:T1=JOYSTK(1)*
3
440 IF J0=T0 AND J1=T1 THEN 300
450 PUT(J0,J1)-(J0+10,J1+12),A1,
PSET
```

FORMA DE EVITAR LA INTERMITENCIA EN LAS FIGURAS

La figura aparece intermitentemente sólo cuando nos movemos. Ahora bien, cuando nos movemos rápidamente comprobaremos que

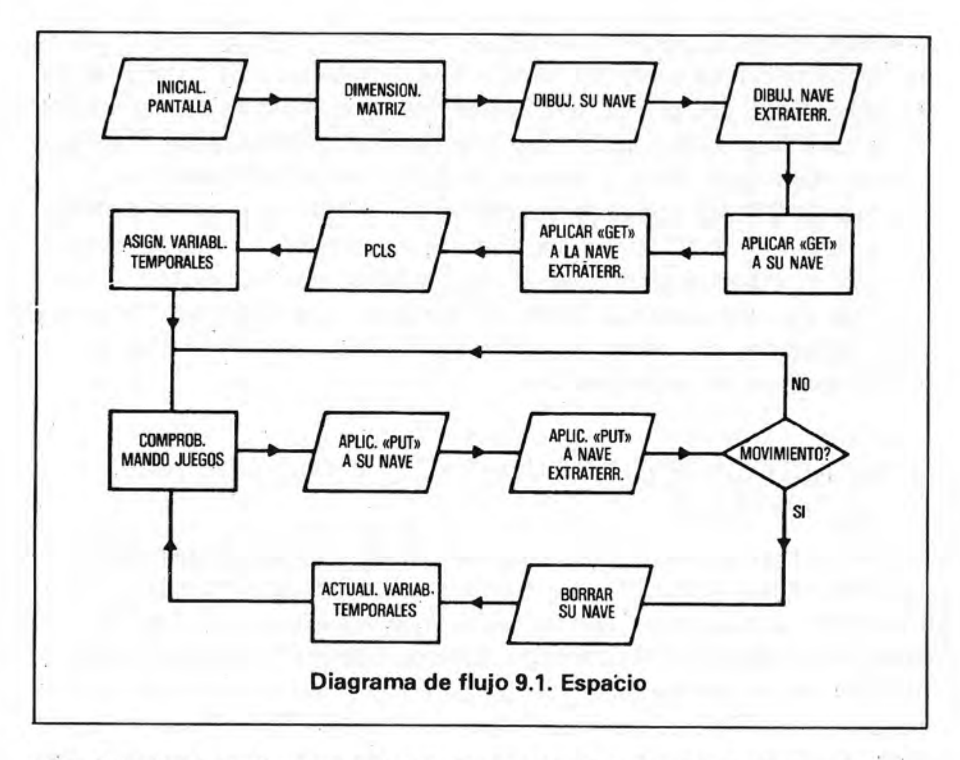


Diagrama de flujo 9.1. Espacio

esto funciona erróneamente puesto que se borra la nueva posición en vez de la anterior. Para resolver este problema tenemos que realizar la instrucción PUT empleando T0 y T1 en vez de J0 y J1. Apliquemos también la misma instrucción a la otra figura con las coordenadas J2 y J3. Como aún no hemos definido las variables J2 y J3, la figura aparecerá en la esquina superior izquierda de la pantalla (0,0).

```
390 PUT(J2,J3)-(J2+21,J3+15),A3,
PSET
450 PUT(T0,T1)-(T0+10,T1+12),A1,
PSET:T0=J0:T1=J1
```

LA SUPERPOSICION DE FIGURAS

Ahora nos podemos mover continuamente y sin brusquedades. De todas maneras, compruébese qué pasa cuando coloca la nave

cuadrada sobre el platillo volante. Al hacer PUT con la nave cuadrada, se borra buena parte del platillo. Cámbiense los PSET del final de las líneas 380 y 390 por un OR. Obsérvese que ahora se puede sobreponer una figura en la parte superior de la otra. Hay algún inconveniente, claro está. En la práctica, la única desventaja reside en que, empleando OR, las dos matrices deben ser comparadas por el sistema antes de hacer PUT con el segundo. Esto requiere un poco más de tiempo, lo cual repercute en el movimiento que no es tan fluido. Cuál de los dos sistemas debe utilizarse en una situación determinada depende, en buena medida, de la importancia relativa de la velocidad y de la superposición.

EL ESTABLECIMIENTO DE CONTACTO ENTRE DOS NAVES EN LA PANTALLA

Ahora que ya podemos mover correctamente dos naves sobre la pantalla, vamos a añadir un poco de acción. Empezaremos por poner en contacto una nave con la otra. Podemos ver qué hay en un espacio por delante del cuadrado que engloba a las figuras, haciendo PPOINT sobre ese espacio frontal. Este punto corresponde al centro del eje X de la figura. Hay que hacer, por tanto, PPOINT (J0 + 5,J1) para cualquier posición en que se encuentre la nave. El borde exterior del platillo es del color 6 y el centro de color 7. Esta circunstancia nos permitirá distinguir entre un contacto suave y otro brusco, que indicaremos con dos sonidos distintos. Esta línea está colocada inmediatamente después de las que colocan las naves en una posición dada pero antes de que las naves sean borradas, se verá permanentemente a las dos naves mientras se oye el sonido que indica el tipo de contacto. Se verán incluso en el caso de que, durante un momento, se vean doble ya que la instrucción PLAY se ejecuta antes que la rutina de borrado (Diagrama de flujo 9.2). Si, en cambio, se decide colocar la instrucción PLAY después de la rutina de borrado, se verá cómo ambas naves desaparecen por un instante. Cuando ya se piense que uno es un experto en esto de poner en contacto ambas naves, inténtese tocar el puntito que hay en medio (color 8).

```
400 IF PPOINT(J0+5,J1)=6 THEN PL
AY"T255O5C" ELSE IF PPOINT(J0+5,
J1)=7 THEN PLAY"T10O1B-"
```

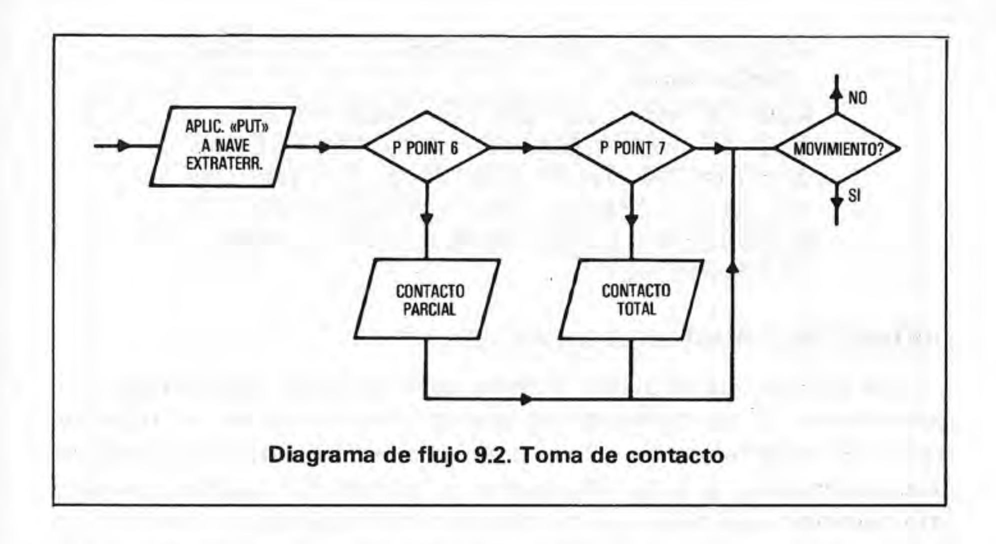


Diagrama de flujo 9.2. Toma de contacto

«FASERS OMNIDIRECCIONALES» EN UN JUEGO ESPACIAL

Si uno se siente un poco más belicoso deseará disponer de potentes fásers omnidireccionales. Es muy fácil hacerlos (línea 430) dibujando, cada vez que se pulse el botón del mando para juegos, una línea con la instrucción LINE desde su posición (J0 + 5,J1) (la punta de su nave) a la del enemigo (J2 + 10,J3 + 7) (el centro de la nave). La línea se dibujará en color cián mediante la instrucción COLOR 6,5. Después se borra con PRESET y se genera el sonido correspondiente al fáser con una instrucción PLAY. Si se quiere limitar el alcance de los fásers, se añade una comprobación para conocer la distancia absoluta total que le separa de sus objetivos sobre los ejes X e Y (línea 410, empléese ABS). Si parece que los fásers omnidireccionales resultan demasiado sencillos de utilizar se añade otra comprobación que sólo permita disparar cuando la nave enemiga se encuentre ante el jugador. Para detectar esta situación se deben comparar los desplazamientos relativos de ambas naves a lo largo del eje X(ABS(J0 — J2)) con una cifra en particular como, por ejemplo, la que hemos empleado en este ejemplo: el 2 (Línea 420) (Figura 9.1). Si se quiere que la limitación comprenda tanto que el enemigo esté situado a una determinada distancia como que se halle frente al jugador, entonces será mejor que se intercambien las líneas 410 y 420 entre sí (Diagrama de flujo 9.3).

```
410 IF ABS(J0-J2)+ABS(J1-J3)>100
 THEN 440
420 IF ABS(J0-J2)>2 THEN 440
430 IF PEEK(65280)=126 OR PEEK(6
5280)=254 THEN COLOR 6,5:LINE(J0
+5,J1)-(J2+10,J3+7),PSET:PLAY"T2
5505DEF+":LINE(J0+5,J1)-(J2+10,J
3+7),PRESET
```

DISEÑO DE OBJETIVOS MOVILES

Nos parece que el platillo volante está sufriendo más castigo del que merece. Es ya momento de que lo convirtamos en un objetivo móvil. Si incrementamos J2 en una cantidad IN, tras cada ciclo de comprobaciones, y luego dibujamos y borramos el platillo volante, obtendremos una nave que se desplaza de izquierda a derecha por la parte superior de la pantalla. Observaremos también una ligera oscilación en ella. Si no queremos que la nave enemiga sufra una avería total, hemos de dar a IN el valor 1 en la línea 10.

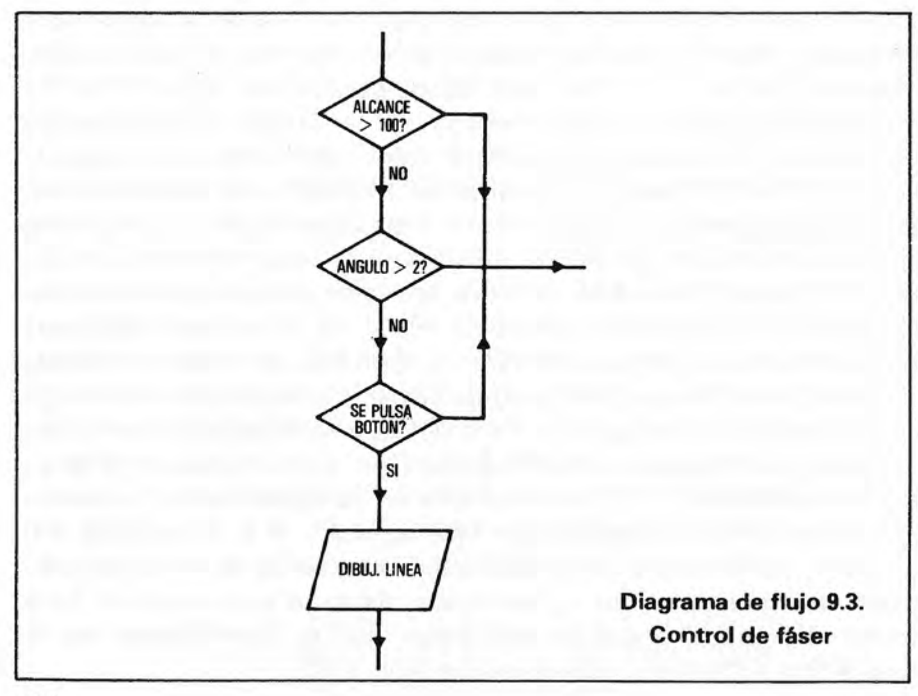


Diagrama de flujo 9.3. Control de fáser

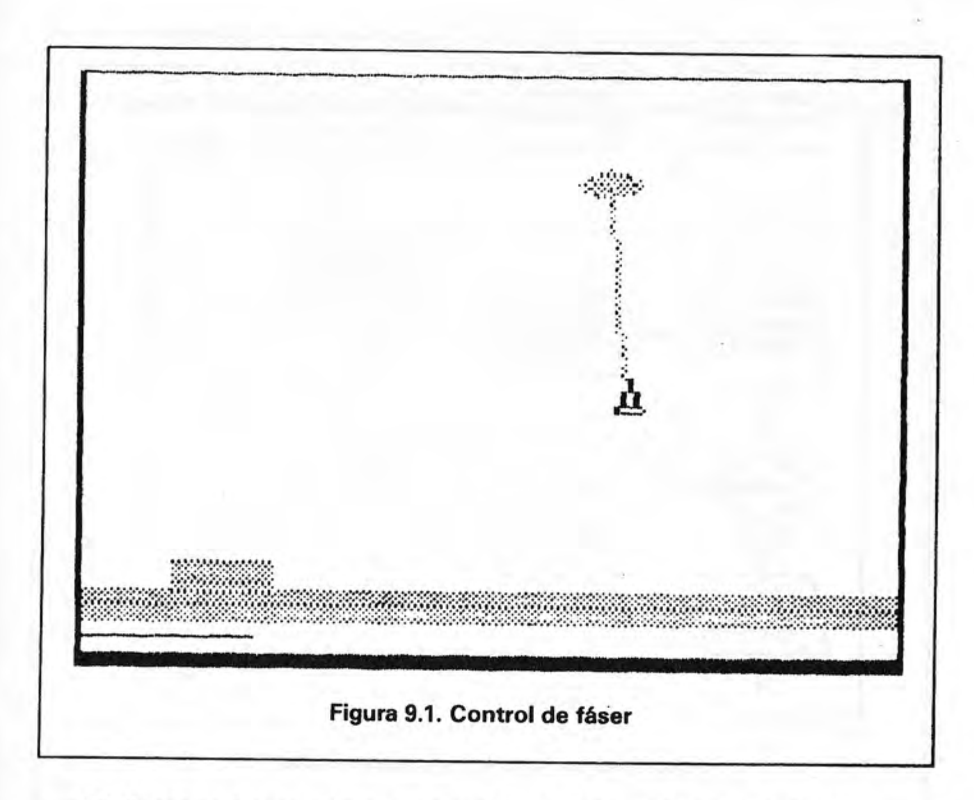


Figura 9.1. Control de fáser

Si damos el valor 0 a J2 cuando se llega a la línea 230, la nave saldrá de la pantalla y reaparecerá por la parte izquierda. Para hacer las cosas cada vez más difíciles, podemos aumentar su desplazamiento cada vez que sale de la pantalla incrementando el valor de IN (Diagrama de flujo 9.4). Si el jugador no puede acertar ahora al enemigo, modifíquese el alcance y ángulo de disparo de los propios fásers. Si, además, se incrementa el valor de J3 al final de la línea 310 el platillo volante pasará cada vez más bajo y a mayor velocidad.

```
10 PMODE 3,1:SCREEN 0,1:PCLS:IN=1
310 J2=J2+IN:IF J2>230 THEN J2=0:
IN=IN+1:J3=J3+10
440 IF J0=T0 AND J1=T1 THEN 460
460 PUT(J2,J3)-(J2+21,J3+15),A1,P
SET.
```

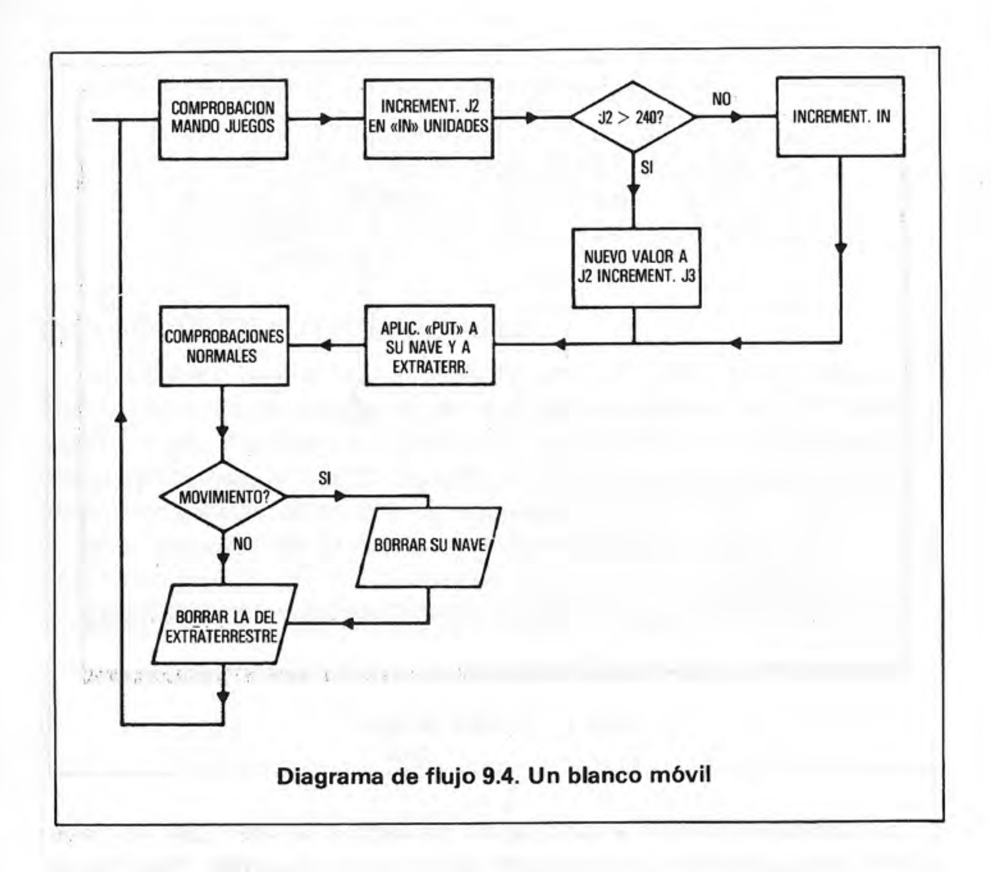


Diagrama de flujo 9.4. Un blanco móvil

DESARROLLO GRAFICO DE ATAQUES EXTRATERRESTRES

Ahora que los extraterrestres se mueven, les ha llegado el momento de pasar a la ofensiva (Diagrama de flujo 9.5). Les daremos también a ellos un fáser omnidireccional de alcance limitado y de tiempo de recarga relativamente largo e imprevisible (dependerá de la variable RT = RND(10)). Si el jugador se encuentra a unos 100 «metrones» de los extraterrestres, hay un 20 % de posibilidades de que le disparen. El jugador tiene la ventaja de disponer de un fáser inagotable y de mayor maniobrabilidad. Por contra, su ángulo de tiro es más estrecho. Por descontado hay que disparar contra ellos y escabullirse. Los dos tipos de fásers (el suyo y el de los enemigos) se distinguen por su color y sonido diferentes.

```
320 RT=RND(10):IF RT>8 AND ABS(J
2-J0)+ABS(J3-J1)<100 THEN LINE(J
2+10,J3+7)-(J0+5,J1),PSET:PLAY"T
150O3FGE":LINE(J2+10,J3+7)-(J0+5
,J1),PRESET
```

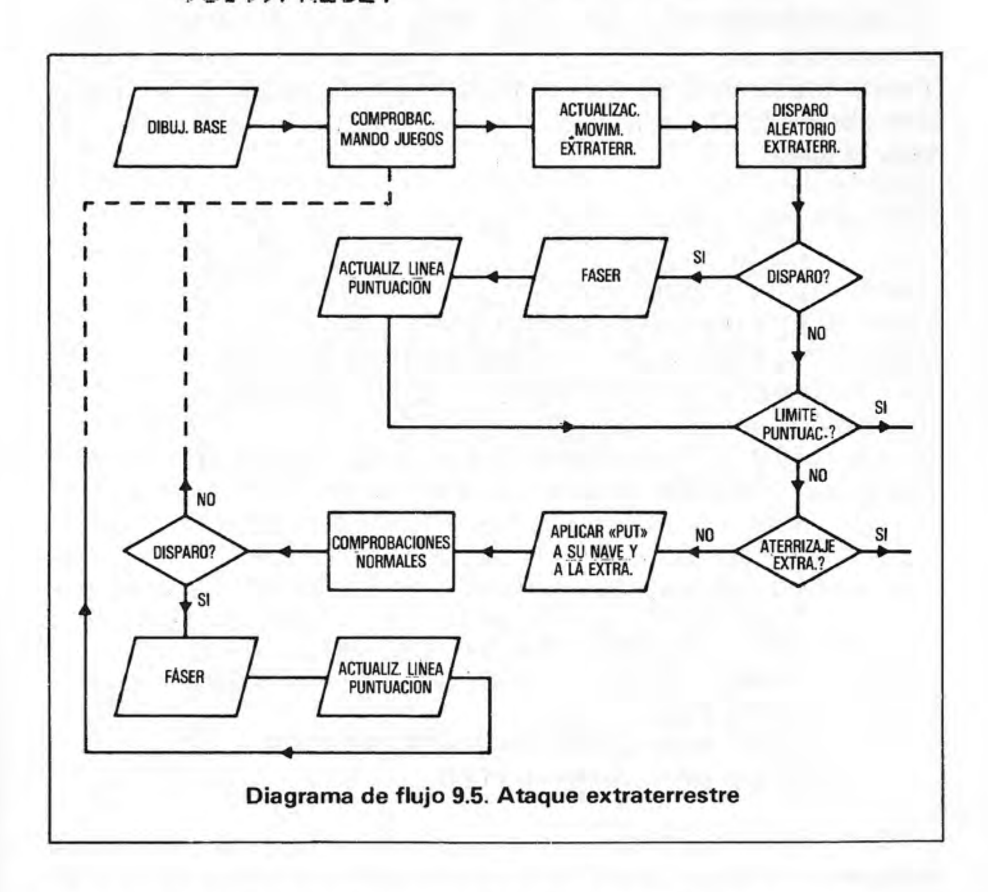


Diagrama de flujo 9.5. Ataque extraterrestre

PRESENTACION DE PUNTUACIONES EN LA PANTALLA

Ha llegado el momento de llevar algún tipo de puntuación para saber quién ha acertado y cuánto daño ha hecho. Como los fásers siempre aciertan sabremos que alguien ha obtenido puntos porque ha utilizado cualquiera de las líneas que se encargan de los fásers. El daño producido debería depender de la distancia al objetivo y, por

tanto, tiene que estar relacionado con la diferencia entre las coordenadas. En cualquier caso, la separación máxima son 100 unidades, así que ¿por qué no dar un punto por cada unidad que nos separe del objetivo? Lo haremos mediante (100 — ABS(J0 — J2) — ABS(J1 — J3)). Su puntuación se guardará en la variable YS y la de los extraterrestres en AS. Una forma sencilla de presentar estas puntuaciones en la pantalla es como líneas de color que atraviesen la parte inferior de la pantalla con una longitud a escala de la puntuación obtenida. Si cualquiera de ambas alcanza el valor 5000, termina el juego.

```
320 RT=RND(10):IF RT>8 AND ABS(J
2-J0)+ABS(J3-J1)<100 THEN COLOR
8,5:LINE(J2+10,J3+7)-(J0+5,J1),P
SET:PLAY"T150O3FGE":LINE(J2+10,J
3+7)-(J0+5,J1),PRESET:AS=AS+(100
-ABS(J0-J2)-ABS(J1-J3)):LINE(0,1
85)-(AS/20,185)),PSET
330 IF AS>5000 THEN 2000 ELSE IF
 YS=5000 THEN 3000
430 IF PEEK(65280)=126 OR PEEK(6
5280)=254 THEN COLOR 6,5:LINE(J0
+5,J1)-(J2+10,J3+7),PSET:PLAY"T2
55O5DEF+":LINE(J0+5,J1)-(J2+10,J
3+7),PRESET:YS=YS+(100-ABS(J0-J2
)-ABS(J1-J3):LINE(0,190)-(YS/20,
190),PSET
2000 REM GANA EXTRATERRESTRE
3000 REM GANA USTED
```

Otra forma que tienen los extraterrestres de ganar es cuando aterrizan en nuestras bases. El trazado de éstas es muy sencillo (240) y es fácil detectarlas comprobando las coordenadas desde la izquierda, ya que los extraterrestres siempre son zurdos.

```
240 COLOR 7,5:LINE(0,170)-(255,1
80),PSET,BF:COLOR 6,5:LINE(30,17
0)-(60,160),PSET,BF
340 IF J2>45 AND J3>145 THEN 400
0
```

```
4000 REM ATERRIZA EL EXTRATERRES
TRE
```

LA OBTENCION DE LOS EFECTOS DE EXPLOSIONES

Como toque final a esta épica batalla espacial nos olvidaremos de nuestro fáser y lo sustituiremos por bombas de fusión que tienen que estallar muy cerca del adversario para ser efectivas (Diagrama de flujo 9.6). Como las bombas de fusión son muy pesadas, la velocidad de nuestra nave se verá reducida en buena medida. Uno acabará por preguntarse si sus escudos de protección podrán resistir durante el tiempo suficiente el fuego enemigo para colocarse en la posición adecuada de disparo.

Utilizaremos ahora el mando para juegos (joystick) como controlador de dirección en vez de controlador de posición como hacíamos antes. Para conseguir esto realizaremos una comprobación lógica respecto a la posición que se ocupa. Una de las ventajas de utilizar un mando para juegos es que uno puede moverse simultáneamente en dos direcciones. Tenemos que comprobar también que no nos hemos salido de la pantalla (para evitar errores FC) o que no nos hemos metido por debajo del nivel de la superficie terrestre. La posición inicial se define dando valores a J0 y J1 en la línea 10. Bórrense la líneas 400 - 420 pues realizan comprobaciones que ahora no son necesarias.

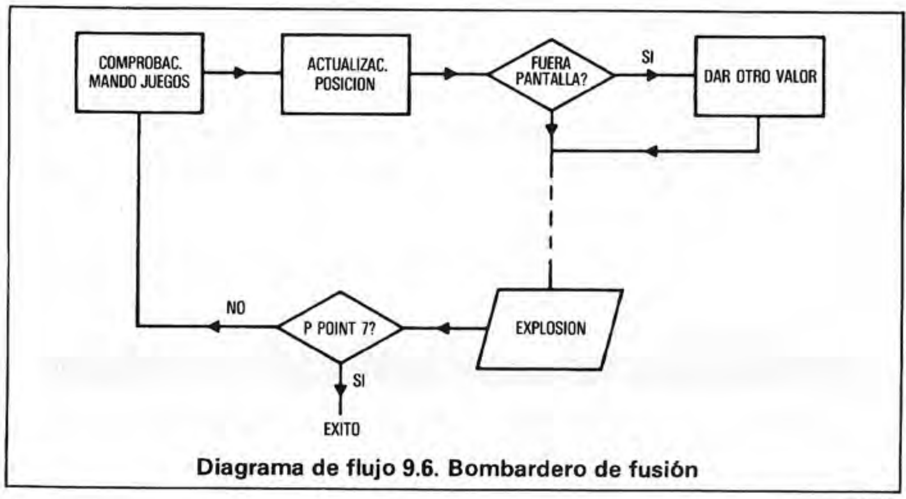


Diagrama de flujo 9.6. Bombardero de fusión

Hemos instalado en la línea 430 las bombas de fusión que sustituyen a los fásers. Cuando el jugador apriete el botón, aparecerá una explosión circular de color 7. Para comprobar si estamos lo suficientemente cerca del enemigo como para destruirle, realizamos una comprobación con PPOINT inmediatamente después de que la explosión llegue a su máximo. Si esto ocurre cuando el color 7 está en las coordenadas (J2 + 5,J3 + 2), que corresponden a la parte central inferior del vehículo extraterrestre (normalmente de color 6) entonces nuestro disparo ha tenido éxito. En caso contrario, la explosión se desvanece (Fig. 9.2).

```
10 PMODE 3,1:SCREEN 0,1:PCLS:IN=
1:J0=128:J1=150
300 J0=J0+((JOYSTK(0)<31)-(JOYST
K(0)>32)):J1=J1+((JOYSTK(1)<31)-
(JOYSTK(1)>32)):IF J0<4 THEN J0=
5 ELSE IF J0>240 THEN J0=239
305 IF J1<4 THEN J1=5 ELSE IF J1
>150 THEN J1=149
```

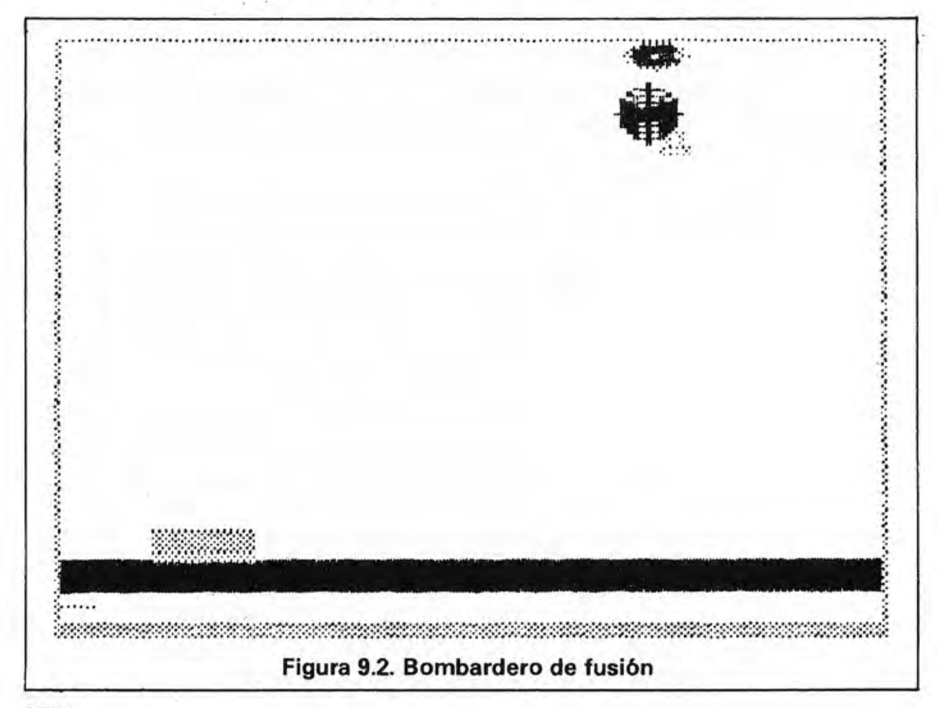


Figura 9.2. Bombardero de fusión

```
430 IF PEEK(65280)=126 OR PEEK(6
5280)=254 THEN FOR N=1 TO 10:CIR
CLE(J0,J1),N,7:NEXT N:IF PPOINT(
J2+5,J3+12)=7 THEN 3000 ELSE FOR
 N=10 TO 1 STEP-1:CIRCLE(J0,J1),
N,5:NEXT N
```

La presentación en pantalla destellea bastante. Esto se debe a la gran cantidad de comprobaciones que hay que llevar a cabo. Afortunadamente existe una manera de hacer que las cosas sean un poco menos discontinuas. Se basa en el hecho de que J0 y J1 sólo pueden cambiar en una unidad cada vez. La solución consiste en cambiar el OR del final de la línea 380 por un PSET. De esta forma, los puntos de la zona del objetivo toman valores que están de acuerdo con los detalles almacenados en la matriz correspondiente.

Sin embargo, cuando nos movamos no realizaremos correctamente el borrado. Hay que hacer PUT y GET con un espacio vacío adicional alrededor de la nave. Así se borrará automáticamente la anterior posición ya que las nuevas y las anteriores se solaparán. Las líneas 440 y 450 deben eliminarse.

```
20 DIM A1(0,9):DIM A2(0,10):DIM
A3(0,9)
200 GET(2,2)-(18,20),A2,G
380 PUT(J0,J1)-(J0+16,J1+19),A2,
PSET
```

Hemos ido ampliando nuestros programas espaciales pero hemos dejado intacta la presentación en pantalla. En la parte final de este capítulo vamos a pasar a un escenario completamente diferente.

LA SIMULACION DE UN ALUNIZAJE

Sin lugar a dudas, uno de los primeros juegos que se jugaron con un ordenador fue una versión rudimentaria del alunizaje. Vamos a ver cómo podemos diseñar una versión gráfica basada en esta misma idea. Volveremos a emplear nuestra nave espacial original y modificaremos el programa del ataque extraterrestre (Diagrama de flujo 9.7).

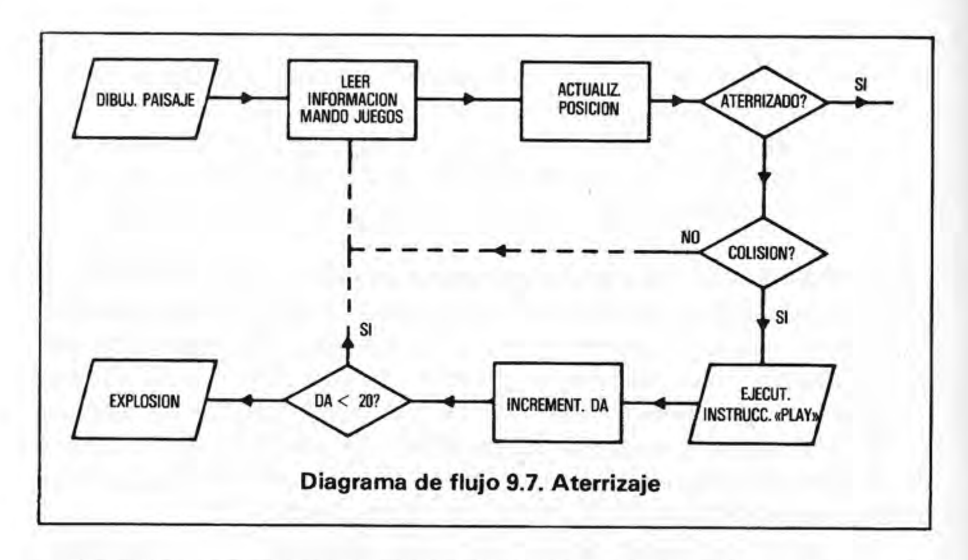


Diagrama de flujo 9.7. Aterrizaje

Lo primero que prepararemos serán zonas en las que no se pueda circular. Colocaremos una en la parte superior de la pantalla y otra en la inferior. Entre las dos, dejaremos un estrecho corredor a través del cual deberemos avanzar para alcanzar nuestra plataforma de aterrizaje. La zona de juego puede definirse fácilmente limitándola con unas cuantas líneas irregulares en la parte superior e inferior y coloreando las distintas zonas que limitan mediante la instrucción PAINT.

LA OBTENCION DE UN SOMBREADO PARA ZONAS RESERVADAS

Si usted ha podido ver algunas de las distintas versiones comerciales de este juego, habrá comprobado que normalmente trabajan en blanco y negro y utilizan un sombreado de líneas rectas para indicar las partes sólidas contra las que no se puede chocar. Es fácil conseguir este efecto. Todo lo que hay que hacer es preparar nuestro dibujo en un modo de cuatro colores y luego observarlo con un modo de dos. El efecto de rayado (Fig. 9.3) se consigue por la forma en que se han codificado los colores. En el modo PMODE 4 cada punto de la pantalla está controlado por tan sólo un bit que únicamente puede estar activado o desactivado. En el modo PMODE 3 los puntos de la pantalla quedan controlados por dos bits y los cuatro colores se codifican así:

Ambos bits desactivados	color 1 ó 5
Bit izquierdo activado	color 2 ó 6
Bit derecho activado	color 3 ó 7
Ambos bits activados	color 4 ó 8

La forma en que se mapean estos bits depende del modo en que se está trabajando en el momento que se les asignan valores. Esto significa que si se colorean objetos en uno de los PMODEs que permiten cuatro colores y luego se pasa a un PMODE que emplee sólo dos, obtendrá rayas oscuras y claras (las oscuras a la derecha de las claras) formadas por los dos colores intermedios. La plataforma de aterrizaje se representa por una línea (a la que se ha aplicado la instrucción PAINT,B) situada a la izquierda en el fondo de un silo de protección un tanto estrecho. Hemos mantenido la instrucción SCREEN de la línea 10 de forma que se pueda ver cómo va apareciendo la figura en la pantalla. Si quiere impresionar a sus amigos, quite esa instrucción: así no se verá nada hasta la instrucción SCREEN que hay después de PMODE 4,1 en la línea 290.

```
10 PMODE 3,1:SCREEN 1,1:PCLS
240 LINE(0,20)-(50,50),PSET:LINE
(50,50)-(60,40),PSET LINE(60,40)
-(120,70),PSET:LINE(120,70)-(120
,120),PSET:LINE(120,120)-(180,13
5),PSET:LINE(180,135)-(255,30),P
SET:PAINT (10,10),6,8
250 LINE(0,100)-(50,100),PSET:LI
NE(50,100)-(60,120),PSET:LINE(60
,120)-(120,150),PSET:LINE(120,15
0)-(190,160),PSET:LINE(190,160)-
(230,110),PSET:LINE(230,110)-(23
0,140),PSET:LINE(230,140)-(245,1
40),PSET:LINE(245,140)-(245,100)
,PSET
260 LINE(245,100)-(255,80),PSET:
PAINT(255,100),6,8:LINE(230,140)
-(245,145),PSET,B:PAINT(235,142)
,7,8
290 PMODE4,1:SCREEN1,1
440 IF J0=T0 AND J1=T1 THEN 300
```

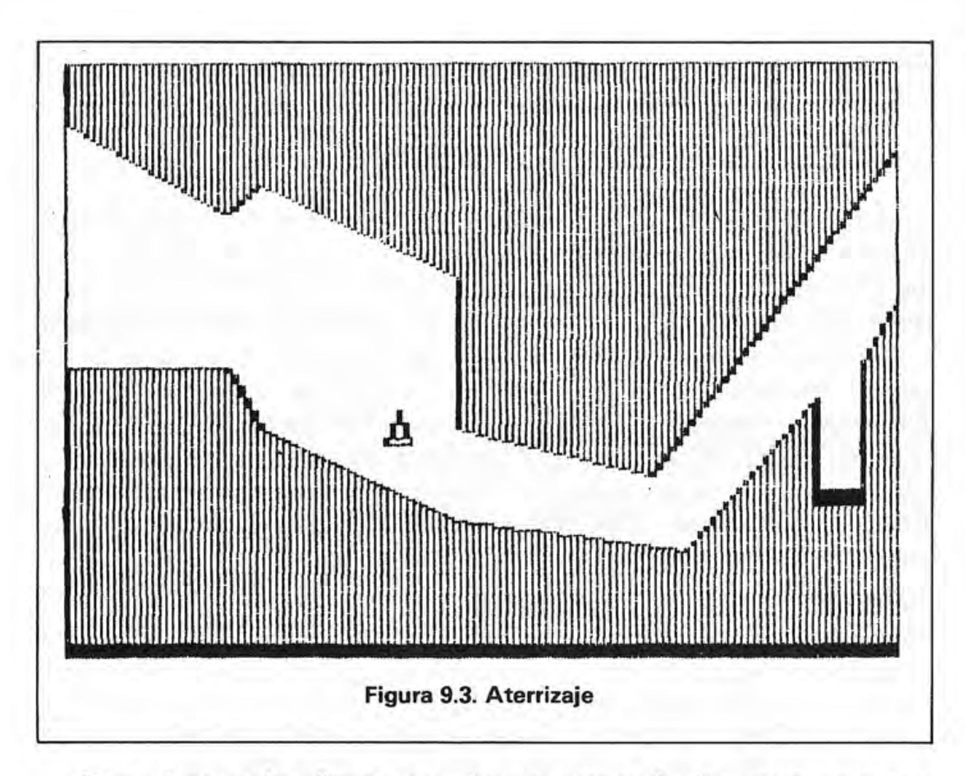


Figura 9.3. Aterrizaje

Si se borran las líneas que se encargan de los extraterrestres (110, 210, 310 - 330, 390, 400 - 430, 460, 2000 y 3000) y luego se añaden las modificaciones que acabamos de sugerir, se comprobará que aún se puede desplazar la nave del jugador por la pantalla mediante el mando para juegos (joystick). Hay que coger práctica en avanzar por el estrecho corredor que se ha dejado. Hay que aprovecharse de que todavía no está penalizado tocar las paredes. Se comprobará que cuando se choca contra la pared lo que ocurre es que ésta resulta borrada. Desde luego nos resulta imprescindible una comprobación que vea si hemos chocado contra la pared o no. Podemos realizarla viendo si cualquiera de las cuatro esquinas de nuestra nave, tras aplicar la instrucción PPOINT, tiene el valor 0.

EFECTOS SONOROS Y GRAFICOS DE UNA EXPLOSION

Hemos de añadir una coordenada adicional a la segunda del eje X de forma que inspeccionemos un punto de la pantalla que

habrá sido activado mediante la instrucción PAINT en modo PMODE 3. Si se detecta una colisión saltamos a una subrutina en la línea 5000 que genera el sonido apropiado y aumenta el valor de los desperfectos sufridos (representados por la variable DA). Esta variable se compara con un límite determinado. Si se rebasa éste, se produce una explosión formada por una serie de círculos concéntricos que se ensanchan progresivamente. Los ruidos de la explosión están en relación con el tamaño de los círculos. La explosión se va desvaneciendo lentamente dejando tras de sí algunos fragmentos de escombros.

```
360 IF PPOINT (J0,J1)>0 OR PPOIN
T(J0+11,J1)>0 OR PPOINT(J0,J1+13
)>0 OR PPOINT(J0+11,J1+13) >0 TH
EN GOSUB 5000
5000 PLAY"O1T255L255CDCDCDGECDCD
C":DA=DA+1:IF DA<20 THEN RETURN
ELSE FOR N=1 TO 15 STEP 2:PLAY"V
"+STR$(N*2)+"CCDD":CIRCLE(J0+5,J
1+6),N,1:NEXT N:FOR N=15 TO 1 ST
EP-1:CIRCLE(J0+5,J1+6),N,0:NEXT
N:RUN
```

Para complicarnos la vida un poco más retiraremos al jugador el control directo y de los motores auxiliares (izquierdo y derecho). También añadiremos el efecto de la gravedad. Las líneas 230 a 440 tienen que borrarse y cambiarse por las siguientes.

```
10 PMODE 3,1:SCREEN 1,1:PCLS:J0=
10:J1=70:T0=J0:T1=J1
300 LR=LR+(JOYSTK(0)<20)-(JOYSTK
(0)>40):UD=UD+(JOYSTK(1)<20)-(JO
YSTK(1)>40)
310 J0=J0+SGN(LR):J1=J1+(2*(SGN(
UD))):J1=J1+1
450 PUT (T0,T1)-(T0+12,T1+12),A1
,PSET:T0=J0:T1=J1
1000 GOTO300
```

CONTROL DE LOS DESPLAZAMIENTOS DE LAS NAVES ESPACIALES

En la línea 300 se incrementan una serie de variables (LR para desplazamiento horizontal y UD para el vertical) según cuál sea la posición del mando para juegos (joystick). Ahora, en vez de dar por resultado una posición, esta línea da un resultado lógico. La primera acción que realiza la línea 310 es añadir el SGN de LR a J0 para provocar un desplazamiento adecuado sobre el eje X. Obsérvese que cuanto más tiempo se haya estado moviéndose en una dirección dada, más tiempo necesitará el signo (SGN) para cambiar. Por ejemplo, si se ha movido hacia la derecha durante tres puntos consecutivos LR será 3 y necesitará cuatro movimientos más en dirección opuesta para que el SGN cambie a menos. El motor principal es más potente que los auxiliares ya que está multiplicado por dos. Por otra parte, la gravedad (J1 = J1 + 1) siempre atrae al jugador hacia abajo. Si desconecta totalmente su motor se dará cuenta de que flotará como un globo lastrado. Las coordenadas iniciales corresponden al borde izquierdo de la pantalla y se fijan en la línea 10.

Lo último que haremos será comprobar si el jugador ha conseguido llegar intacto a su destino. Esto se hace con una sencilla comprobación que compara sus coordenadas con las de la plataforma de aterrizaje.

```
310 J0=J0+SGN(LR):J1=J1+(2*(SGN(
UD))):J1=J1+1:IF J0>228 AND J0<2
32 AND J1>128 AND J1<132 THEN 60
00
6000 PRINT "EXITO"
```

AHORA ES SU TURNO

Las modificaciones que puede usted hacer sobre estas ideas son infinitas pero como hemos dejado las rutinas finales algo incompletas creemos que deberá usted empezar a trabajar por ahí. Aquí van unas cuantas sugerencias:

1) Vuelva a diseñar la nave de acuerdo con sus propias especificaciones. Asegúrese de que modifica las instrucciones DIM, GET y PUT correspondientes.

2) Prepare un juego en el que los extraterrestres puedan aparecer súbitamente en cualquier punto de la pantalla y dejen caer una nube de veneno de la que usted sólo pueda escapar por su velocidad.
3) Prepare una versión para dos jugadores de «Ataque extraterrestre». Cada uno de ellos debe poder controlar su nave independientemente.
4) Introduzca los efectos aleatorios del viento solar en las rutinas de aterrizaje y toma de contacto.
5) Compruebe la velocidad (SGN(UD)) con la cual aterriza y establezca niveles de habilidad o dificultad asociando a cada uno de ellos un corredor tanto más estrecho cuanto más alto sea el nivel.
6) Prepare una versión del «Aterrizaje» para dos jugadores en la que la pantalla quede dividida en dos mitades idénticas cada una de las cuales estará controlada independientemente de forma que ambos jugadores puedan competir a ver quién es el que llega a su plataforma de aterrizaje antes (Fig. 9.4).

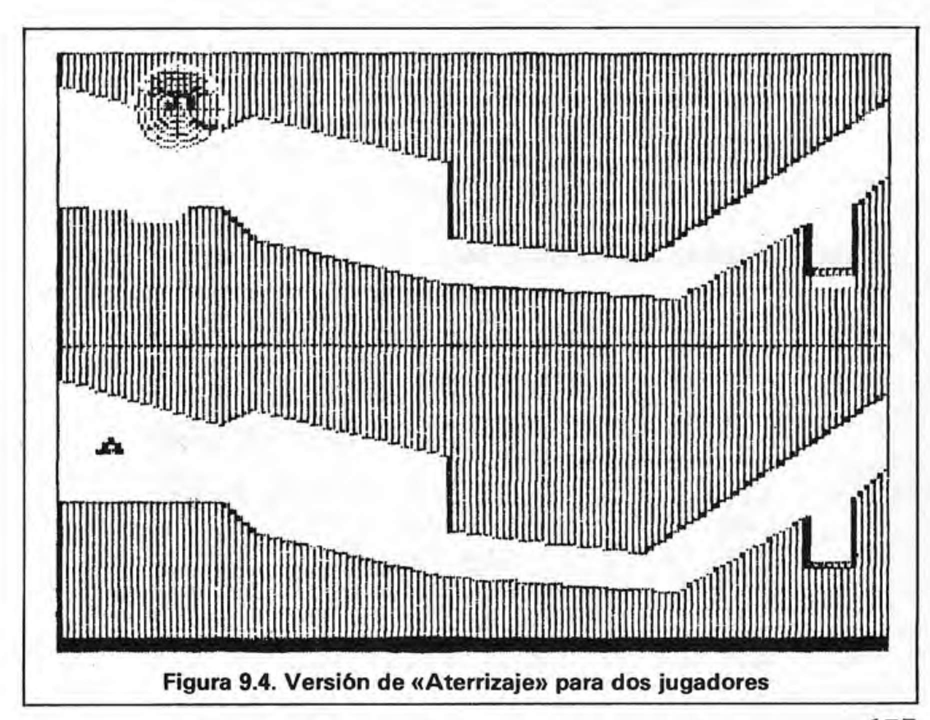


Figura 9.4. Versión de «Aterrizaje» para dos jugadores

Listado completo de «Espacio»

```
10 PMODE 3,1:SCREEN 1,1:PCLS
20 DIM A1(0,9):DIM A2(0,3):DIM A
3(0,9)
100 DRAW"C8BM9,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
110 CIRCLE(40,10),10,6,0.5:PAINT
(40,10),7,6:PSET(40,10,8)
200 GET(4,4)-(14,16),A2,G
210 GET(31,5)-(52,20),A3,G
220 PCLS
230 T0=JOYSTK(0)*4:T1=JOYSTK(1)*
3
300 J0=JOYSTK(0)*4:J1=JOYSTK(1)*
3
380 PUT(J0,J1)-(J0+10,J1+12),A2,
OR
390 PUT(J2,J3)-(J2+21,J3+15),A3,
OR
440 IF J0=T0 AND J1=T1 THEN 300
450 PUT (T0,T1)-(T0+10,T1+12),A1
,PSET:T0=J0:T1=J1
1000 GOTO 300
```

Listado completo de «Contacto»

```
10 PMODE 3,1:SCREEN 1,1:PCLS
20 DIM A1(0,9):DIM A2(0,3):DIM A
3(0,9)
100 DRAW"C8BM9,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
110 CIRCLE(40,10),10,6,0.5:PAINT
(40,10),7,6:PSET(40,10,5)
200 GET(4,4)-(14,16),A2,G
210 GET(31,5)-(52,20),A3,G
220 PCLS
230 T0=JOYSTK(0)*4:T1=JOYSTK(1)*
3
```

```
300 J0=JOYSTK(0)*4:J1=JOYSTK(1)*
3
380 PUT(J0,J1)-(J0+10,J1+12),A2,
OR
390 PUT(J2,J3)-(J2+21,J3+15),A3,
OR
400 IF PPOINT(J0+5,J1)=6 THEN PL
AY"T25505C" ELSE IF PPOINT(J0+5,
J1)=7 THEN PLAY"T1001B-"
440 IF J0=T0 AND J1=T1 THEN 300
450 PUT (T0,T1)-(T0+10,T1+12),A1
,PSET:T0=J0:T1=J1
1000 GOTO 300
```

Listado completo de «Control de fásers»

```
10 PMODE 3,1:SCREEN 1,1:PCLS
20 DIM A1(0,9):DIM A2(0,3):DIM A
3(0,9)
100 DRAW"C8BM8,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
110 CIRCLE(40,10),10,6,0.5:PAINT
(40,10),7,6:PSET(40,10,5)
200 GET(4,4)-(14,16),A2,G
210 GET(31,5)-(52,20),A3,G
220 PCLS
230 T0=JOYSTK(0)*4:T1=JOYSTK(1)*
3
300 J0=JOYSTK(0)*4:J1=JOYSTK(1)*
3
380 PUT(J0,J1)-(J0+10,J1+12),A2,
OR
390 PUT(J2,J3)-(J2+21,J3+15),A3,
OR
400 IF PPOINT(J0+5,J1)=6 THEN PL
AY"T25505C" ELSE IF PPOINT(J0+5,
J1)=7 THEN PLAY"T1001B-"
410 IF ABS(J0-J2)+ABS(J1-J3)>100
 THEN 440
```

```
420 IF ABS(J0-J2)>2 THEN 440
430 IF PEEK(65280)=126 OR PEEK(6
5280)=254 THEN COLOR 6,5:LINE(J0
+5,J1)-(J2+10,J3+7),PSET:PLAY"T2
5505DEF+":LINE(J0+5,J1)-(J2+10,J
3+7),PRESET
440 IF J0=T0 AND J1=T1 THEN 300
450 PUT (T0,T1)-(T0+10,T1+12),A1
,PSET:T0=J0:T1=J1
1000 GOTO 300
```

Listado completo de «Un objetivo móvil»

```
10 PMODE 3,1:SCREEN 1,1:PCLS:IN=
1
20 DIM A1(0,9):DIM A2(0,3):DIM A
3(0,9)
100 DRAW"C8BM8,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
110 CIRCLE(40,10),10,6,0.5:PAINT
(40,10),7,6:PSET(40,10,5)
200 GET(4,4)-(14,16),A2,G
210 GET(31,5)-(52,20),A3,G
220 PCLS
230 T0=JOYSTK(0)*4:T1=JOYSTK(1)*
3
300 J0=JOYSTK(0)*4:J1=JOYSTK(1)*
3
310 J2=J2+IN:IF J2>230 THEN J2=0
:IN=IN+1:J3=J3+10
380 PUT(J0,J1)-(J0+10,J1+12),A2,
OR
390 PUT(J2,J3)-(J2+21,J3+15),A3,
OR
400 IF PPOINT(J0+5,J1)=6 THEN PL
AY"T25505C" ELSE IF PPOINT(J0+5,
J1)=7 THEN PLAY"T1001B-"
410 IF ABS(J0-J2)+ABS(J1-J3)>100
 THEN 440
```

```
420 IF ABS(J0-J2)>2 THEN 440
430 IF PEEK(65280)=126 OR PEEK(6
5280)=254 THEN COLOR 6,5:LINE(J0
+5,J1)-(J2+10,J3+7),PSET:PLAY"T2
55O5DEF+":LINE(J0+5,J1)-(J2+10,J
3+7),PRESET
440 IF J0=T0 AND J1=T1 THEN 460
450 PUT (T0,T1)-(T0+10,T1+12),A1
,PSET:T0=J0:T1=J1
460 PUT(J2,J3)-(J2+21,J3+15),A1,
PSET
1000 GOTO 300
```

Listado completo de «Ataque extraterrestre»

```
10 PMODE 3,1:SCREEN 1,1:PCLS:IN=
1
20 DIM A1(0,9):DIM A2(0,3):DIM A
3(0,9)
100 DRAW"C8BM9,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
110 CIRCLE(40,10),10,6,0.5:PAINT
(40,10),7,6:PSET(40,10,5)
200 GET(4,4)-(14,16),A2,G
210 GET(31,5)-(52,20),A3,G
220 PCLS
230 T0=JOYSTK(0)*4:T1=JOYSTK(1)*
3
240 COLOR 7,5:LINE(0,170)-(255,1
80),PSET,BF:COLOR 6,5:LINE(30,17
0)-(60,160),PSET,BF
300 J0=JOYSTK(0)*4:J1=JOYSTK(1)*
3
310 J2=J2+IN:IF J2>230 THEN J2=0
:IN=IN+1:J3=J3+10
320 RT=RND(10):IF RT>9 AND ABS(J
2-J0)+ABS(J3-J1)<100 THEN COLOR
8,5:LINE(J2+10,J3+7)-(J0+5,J1),P
SET:PLAY"T150O3FGE":LINE(J2+10,J
```

```
3+7)-(J0+5,J1),PRESET:AS=AS+(100
-ABS(J0-J2)-ABS(J1-J3)):LINE(0,1
85)-(AS/20,185),PSET
330 IF AS>5000 THEN 2000 ELSE IF
 YS=5000 THEN 3000
340 IF J2>45 AND J3>145 THEN 400
0
380 PUT(J0,J1)-(J0+10,J1+12),A2,
OR
390 PUT(J2,J3)-(J2+21,J3+15),A3,
OR
400 IF PPOINT(J0+5,J1)=6 THEN PL
AY"T25505C" ELSE IF PPOINT(J0+5,
J1)=7 THEN PLAY"T1001B-"
410 IF ABS(J0-J2)+ABS(J1-J3)>100
 THEN 440
420 IF ABS(J0-J2)>2 THEN 440
430 IF PEEK(65280)=126 OR PEEK(6
5280)=254 THEN COLOR 6,5:LINE(J0
+5,J1)-(J2+10,J3+7),PSET:PLAY"T2
5505DEF+":LINE(J0+5,J1)-(J2+10,J
3+7),PRESET:YS=YS+(100-ABS(J0-J2
)-ABS(J1-J3)):LINE(0,190)-(YS/20
,190),PSET
440 IF J0=T0 AND J1=T1 THEN 460
450 PUT (T0,T1)-(T0+10,T1+12),A1
,PSET:T0=J0:T1=J1
460 PUT(J2,J3)-(J2+21,J3+15),A1,
PSET
1000 GOTO 300
2000 REM GANA EXTRATERRESTRE
3000 REM GANA USTED
4000 REM EL EXTRATERRESTRE HA AT
ERRIZADO
```

Listado completo de «Bombardero de fusión»

```
10 PMODE 3,1:SCREEN 1,1:PCLS:IN=
1:J0=128:J1=150
20 DIM A1(0,9):DIM A2(0,10):DIM
A3(0,9)
```

```
100 DRAW"C8BM8,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
110 CIRCLE(40,10),10,6,0.5:PAINT
(40,10),7,6:PSET(40,10,5)
200 GET(2,2)-(18,20),A2,G
210 GET(31,5)-(52,20),A3,G
220 PCLS
230 T0=JOYSTK(0)*4:T1=JOYSTK(1)*
3
240 COLOR 7,5:LINE(0,170)-(255,1
80),PSET,BF:COLOR 6,5:LINE(30,17
0)-(60,160),PSET,BF
300 J0=J0+((JOYSTK(0)<31)-(JOYST
K(0)>32)):J1=J1+((JOYSTK(1)<31)-
(JOYSTK(1)>32)):IF J0<4 THEN J0=
5 ELSE IF J0>240 THEN J0=239
305 IF J1<4 THEN J0=5 ELSE IF J1
>150 THEN J1=149
310 J2=J2+IN:IF J2>230 THEN J2=0
:IN=IN+1:J3=J3+10
320 RT=RND(10):IF RT>8 AND ABS(J
2-J0)+ABS(J3-J1)<100 THEN COLOR
8,5:LINE(J2+10,J3+7)-(J0+5,J1),P
SET:PLAY"T150O3FGE":LINE(J2+10,J
3+7)-(J0+5,J1),PRESET:AS=AS+(100
-ABS(J0-J2)-ABS(J1-J3)):LINE(0,1
85)-(AS/20,185),PSET
330 IF AS>5000 THEN 2000 ELSE IF
 YS=5000 THEN 3000
340 IF J2>45 AND J3>145 THEN 400
0
380 PUT(J0,J1)-(J0+16,J1+18),A2,
PSET
390 PUT(J2,J3)-(J2+21,J3+15),A3,
OR
430 IF PEEK(65280)=126 OR PEEK(6
5280)=254 THEN FOR N=1 TO 10:CIR
CLE(J0,J1),N,7:NEXT N:IF PPOINT(
J2+5,J3+12)=7 THEN 3000 ELSE FOR
 N=10 TO 1 STEP-1:CIRCLE(J0,J1),
N,5:NEXT N
```

```
460 PUT(J2,J3)-(J2+21,J3+15),A1,
PSET
1000 GOTO 300
2000 REM GANA EXTRATERRESTRE
3000 REM GANA USTED
4000 REM EL EXTRATERRESTRE HA AT
ERRIZADO
```

Listado completo de «Aterrizaje»

```
10 PMODE 3,1:SCREEN 1,1:PCLS:J0=
10:J1=70:T0=J0:T1=J1
20 DIM A1(0,9):DIM A2(0,3):DIM A
3(0,9)
100 DRAW"C8BM8,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
200 GET(4,4)-(14,16),A2,G
220 PCLS
240 LINE(0,20)-(50,50),PSET:LINE
(50,50)-(60,40),PSET:LINE(60,40)
-(120,70),PSET:LINE(120,70)-(120
,120),PSET:LINE(120,120)-(180,13
5),PSET:LINE(180,135)-(255,30),P
SET:PAINT(10,10),6,8
250 LINE(0,100)-(50,100),PSET:LI
NE(50,100)-(60,120),PSET:LINE(60
,120)-(120,150),PSET:LINE(120,15
0)-(190,160),PSET:LINE(190,160)-
(230,110),PSET:LINE(230,110)-(23
0,140),PSET:LINE(230,140)-(245,1
40),PSET:LINE(245,140)-(245,100)
,PSET
260 LINE(245,100)-(255,80),PSET:
PAINT(255,100),6,8:LINE(230,140)
-(245,145),PSET,B:PAINT(235,142)
,7,8
290 PMODE 4,1:SCREEN 1,1
300 LR=LR+(JOYSTK(0)<20)-(JOYSTK
(0)>40):UD=UD+(JOYSTK(1)<20)-(JO
YSTK(1)>40)
```

```
310 J0=J0+SGN(LR):J1=J1+(2*(SGN(
UD))):J1=J1+1:IF J0>228 AND J0<2
32 AND J1>128 AND J1<132 THEN 60
00
360 IF PPOINT(J0,J1)>0 OR PPOINT
(J0+11,J1)>0 OR PPOINT(J0,J1+13)
>0 OR PPOINT(J0+11,J1+13)>0 THEN
 GOSUB 5000
370 PUT(T0,T1)-(T0+12,T1+12),A1,
PSET:T0=J0:T1=J1
380 PUT(J0,J1)-(J0+10,J1+12),A2,
OR
1000 GOTO 300
5000 PLAY"O1T255L255CDCDCDGECDCD
C":DA=DA+1:IF DA<20 THEN RETURN
ELSE FOR N=1 TO 15 STEP 2:PLAY"V
"+STR$(N*2)+"CCDD":CIRCLE(J0+5,J
1+6),N,1:NEXT N:FOR N=15 TO 1 ST
EP -1:CIRCLE(J0+5,J1+6),N,0:NEXT
 N:RUN
6000 PRINT "EXITO"
```



Capítulo X

LA SIMULACION DE OPERACIONES FINANCIERAS

Como parece ser que la mayor parte de la gente cree que, con sólo un poco más de suerte, se harían millonarios con cualquier negocio, los juegos de finanzas de todo tipo se han hecho muy populares. Por tanto, vamos a ver cómo tocar los aspectos económicos de nuestra vida con el Dragón.

CARACTERISTICAS DE LOS PROGRAMAS SOBRE INVERSIONES

Consideremos primero una simple inversión. Por descontado se necesita un capital inicial que se guardará en la cuenta bancaria como BK del jugador. Debería haber también un elemento temporal y, por tanto, vamos a darle un valor inicial al reloj. Como la rutina inicial sólo se utiliza una vez en cada partida y como el programa acabará siendo bastante complejo, vamos a colocar esta parte al final del programa. Llamaremos a esta subrutina nada más empezar el programa.

```
10 GOSUB 50000
50000 CLS:BK=10000:TIMER=0:RETUR
N
```

Para informar al jugador de lo que tiene en el banco en un momento dado, tenemos que imprimir tanto un mensaje de información como la cantidad de que disponga. Como más adelante diseñaremos un formato fijo de presentación en pantalla en el que únicamente

se actualizarán las cifras, vamos a poner el mensaje de información en la rutina inicial y la cantidad de dinero en otra línea de información aparte.

```
20 PRINT @ 178,BK;
50000 CLS:BK=10000:TIMER=0:PRINT
 @ 161,"BALANCE BANCARIO";:RETUR
N
```

Ahora hemos de decidir cuánto deseamos invertir (IV) y restar esta cantidad de nuestra cuenta. Si nuestro resultado es negativo llamaremos a la subrutina que hace las veces de director del banco para que nos señale el error que hemos cometido.

```
50 PRINT @ 0,"CUANTO QUIERE INVE
RTIR","";
60 PRINT @ 55,"";:INPUT IV
70 IF BK-IV<0 THEN 40000 ELSE B
K=BK-IV
40000 PRINT @ 0,"NO TIENE USTED
ESA","CANTIDAD EN SU CUENTA SE/O
R":SCREEN 0,1:SOUND 10,10:GOTO 2
0
```

Todos los mensajes aparecen en las dos líneas superiores de la pantalla. Las ,""; al final de la línea 50 nos aseguran que la segunda línea de la pantalla quedará borrada cuando aparezca este mensaje de tan sólo una línea. La instrucción SCREEN 0,1 cambia la combinación de colores de la pantalla por otra que está formada por rojo sobre naranja y que indica claramente cuándo se encuentra el jugador en números rojos. La combinación de colores vuelve a ser la normal (negro sobre verde) en cuanto se vuelve a ejecutar una instrucción PRINT (a menos que se coloque la instrucción SCREEN 0,1 inmediatamente después de la instrucción PRINT). También hemos añadido sonido para llamar la atención del jugador y darle tiempo a leer el mensaje.

La inversión total (TI) será la suma de inversiones parciales (IV) que el jugador realice cada vez.

```
80 TI=TI+IV
```

La rentabilidad neta (NI) de estas inversiones dependerá de los ingresos (IN) y de los gastos (EX) que, a su vez, dependen del promedio de ingresos (IR) y del promedio de gastos (ER). En una inversión totalmente segura, los ingresos siempre serán mayores que los gastos pero, por contra, los tipos de interés serán bajos. Vamos a fijar el promedio de ingresos en un valor del 2 % multiplicando la inversión total por 0.02. Al promedio de gastos le damos el valor 1 % multiplicándolos por 0.01. Esto dará una rentabilidad neta equivalente al 1 % de la inversión.

```
90 IN=TI*0.02:EX=TI*0.01
100 NI=IN-EX
```

La rentabilidad (NI) debe añadirse ahora a la cuenta bancaria (BK) del jugador. Su riqueza total (TW) tiene que actualizarse sumando el contenido de su cuenta bancaria a su inversión total (TI). Entonces tendremos que volver a tomar una decisión acerca de cuánto vamos a invertir.

```
110 BK=BK+NI
120 TW=BK+TI
200 GOTO 20
```

Si ejecuta este programa podrá ver que el balance de su cuenta va aumentando lentamente incluso si usted no invierte dinero alguno.

PRESENTACION EN PANTALLA DE LA MARCHA DE UN NEGOCIO

Sería todo mucho más interesante si dispusiéramos de una presentación en pantalla más completa que nos indicará cómo cambia cada una de las cifras que contribuyen al total. Por tanto, añadiremos:

```
20 PRINT @ 111,IN;:PRINT @ 143;E
X;:PRINT @ 175,BK;:PRINT @ 207,T
I;:PRINT @ 239,TW;
50000 CLS:BK=10000:TIMER=0:PRINT
 @ 97,"RENTA";:PRINT @ 129,"GAST
```

```
OS";:PRINT @ 161,"BALANCE BANCAR
IO";:PRINT @ 193,"OTROS ACTIVOS"
;:PRINT @ 225,"TOTAL ACTIVOS";:R
ETURN
```

Ahora que ya se puede ver cómo se mueve el dinero, es el momento de que volvamos a las duras realidades de la vida en la que los beneficios y los gastos son mucho menos previsibles. Multipliquemos por RND(0) los porcentajes de interés y de gastos que hemos fijado tan alegremente y obtendremos unos nuevos porcentajes que oscilarán entre el 0 y el 100 %. Substitúyase ambos porcentajes por (RND(0)*0.1) y obsérvese el efecto.

```
90 IN=TI*(RND(0)*0.1):EX=TI*(RND
(0)*0.1)
```

Bueno, así no va a ganar nunca a este juego. Será mejor que cambie el porcentaje de gastos por (RND(0)*0.05) para tener alguna esperanza de sobrevivir. Será mejor que añada también una instrucción que compruebe si nos hemos convertido en millonarios o no. Para ello, comprobaremos si nuestra riqueza total (TW) es superior a la riqueza que habíamos tomado como objetivo (TG). Para saber si estamos en bancarrota comprobaremos si la riqueza total es inferior a nuestro límite de crédito (CL) (Diagrama de flujo 10.1).

```
20 PRINT @ 111,IN;:PRINT @ 143;E
X;:PRINT @ 175,BK;:PRINT @ 207,T
I;:PRINT @ 239,TW;:IF TW>TG THEN
 30000 ELSE IF TW<CL THEN 20000
20000 CLS:PRINT @ 72,"ESTA ARRUI
NADO";:PRINT @ 136,"DEBE AL BANC
O",BK;:PRINT @ 260,"PERO SOLO DI
SPONE DE",TI;:FOR N=255 TO 1 STE
P -5:SOUND N,1:NEXT N
20010 PRINT @ 384,"SE ESCAPA A S
URAMERICA";:SOUND 250,5:PRINT @
450,"O QUIERE INTENTARLO OTRA VE
Z?";:SCREEN 0,1:Q$=INKEY$:IF Q$=
"" THEN 20010 ELSE RUN
```

```
30000 CLS:PRINT @ 233,"FELICIDAD
ES":PRINT @ 292,"CONSIGUIO SU OB
JETIVO EN";:PRINT @ 361,TIMER/50
;"SEGUNDOS";:FOR N=1 TO 255 STEP
 5:SOUND N,1:NEXT N:RUN
50000 CLS:BK=10000:TIMER=0:TG=10
00000:CL=100000:PRINT @ 97,"RENT
A";:PRINT @ 129,"GASTOS";:PRINT
@ 161,"BALANCE BANCARIO";:PRINT
@ 193,"OTROS ACTIVOS";:PRINT @ 2
25,"TOTAL ACTIVOS";:RETURN
```

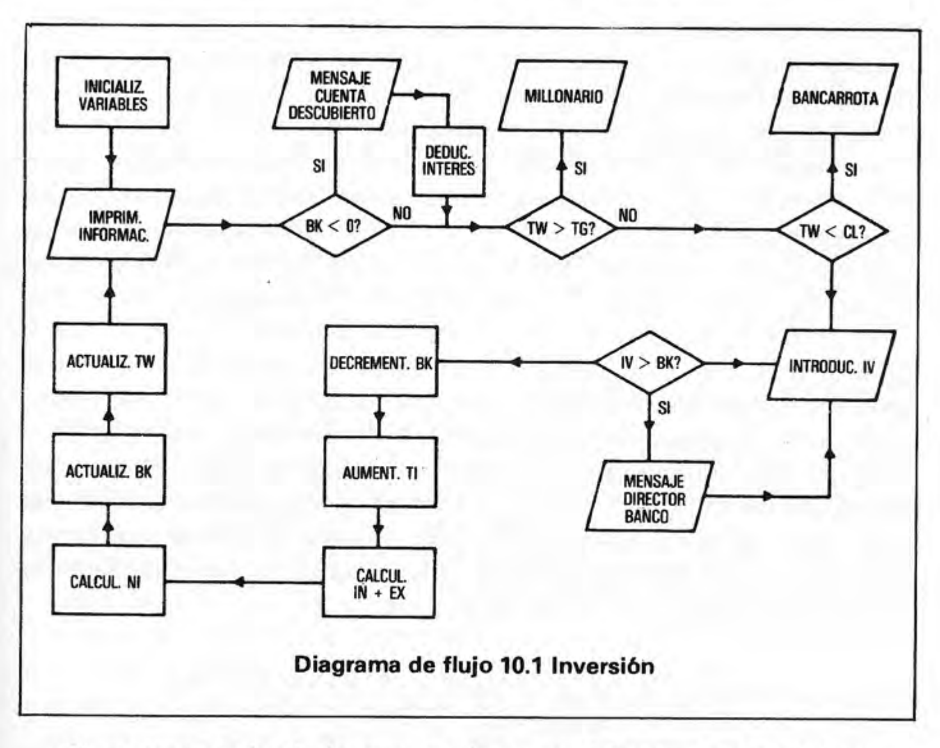


Diagrama de flujo 10.1 Inversión

Si se gasta más de lo que se tiene en el banco se recibirá como penalización un porcentaje de interés poco remunerador lo que provocará su caída en la bancarrota si no se recupera pronto.

```
20 PRINT @ 111,IN;:PRINT @ 143;E
X;:PRINT @ 175,BK;:PRINT @ 207,T
```

```
I;:PRINT @ 239,TW;:IF TW>TG THEN
 30000 ELSE IF TW<CL THEN 20000
ELSE IF BK<0 THEN GOSUB 41000
41000 FORN=1 TO 10:PRINT @ 0,"SU
 CUENTA ESTA EN DESCUBIERTO!!!",
"TIENE QUE PAGAR UN INTERES DEL
20%":SOUND 1,2:NEXT N:BK=BK-(ABS
(BK)*0.2):RETURN
```

GESTION SIMULADA DE UNA CARTERA DE VALORES

Un factor importante a tener en cuenta es en qué invertimos nuestro dinero. Como todos estamos, presumiblemente, convencidos de que hay dinero en el mercado de los microordenadores, vamos a ofrecerle una amplia selección de empresas de fabricación de microordenadores en las que invertir su dinero (cualquier parecido entre estos nombres y el de cualquier compañía existente o desaparecida es puramente accidental). La mejor forma de gestionar nuestras acciones es poner en instrucciones DATA el nombre de las compañías y de los productos que fabrican. Cuando empiece el programa, toda esta información se podrá leer por medio de una instrucción READ y pasar a matrices de cadenas C$(N) (para las compañías) y P$(N) (para los productos) imprimiéndolos uno debajo del otro. Para conseguir esto multiplicaremos la posición de impresión por (32*N) y numeraremos cada compañía imprimiendo N al principio de su línea (Diagrama de flujo 10.2). Si no se ha dado cuenta del cambio del color de fondo por el color 2 (amarillo) y la inclusión de unos cuantos espacios, vuelva atrás y cámbielos pues resulta mucho más elegante imprimir en verde sobre un fondo amarillo.

```
50000 CLS2:BK=10000:TIMER=0:TG=1
000000:CL=100000:PRINT @ 97,"REN
TA";:PRINT @ 129,"GASTOS";:PRINT
 @ 161,"BALANCE BANCARIO";:PRINT
 @ 193,"OTROS ACTIVOS";:PRINT @
225,"TOTAL ACTIVOS";
50010 DIM C$(6):DIM P$(6):FOR N=
1 TO 6:READ C$(N),P$(N):PRINT @
```

```
(257+(32*N)),N;C$(N);:NEXT N:RET
URN
60000 DATA CLEARSIN,EXPECTUM,ACH
RON,ILLUSION,COMPUTERS,JYNX,AREK
 INT'L,AREK 1.6 K,GRANDO,DOGRAN
23,TV PM,MODELO T
```

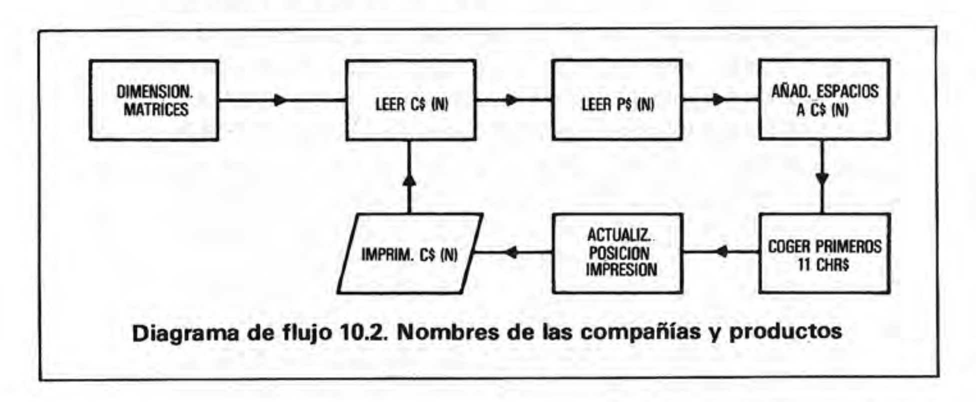


Diagrama de flujo 10.2. Nombres de las compañías y productos

Otra forma sencilla de mejorar la presentación consiste en alinear las cifras mediante PRINT @USING. Si empleamos "$ # # # # # # . # # +" para las variables de la línea que nos informa acerca de nuestra situación, obtendremos un signo $, seis cifras antes de la coma decimal, dos más tras ella y un signo que nos indicará si se trata de un asiento de activo o pasivo.

Cuesta un poco conseguir que los nombres de las compañías queden alineados por la derecha puesto que sus longitudes varían de 5 a 11 caracteres. Un formato para cadena de "%   %" (once espacios) limitará la longitud del nombre de las compañías a once caracteres. Ahora bien, aquellas compañías con un nombre de menos de once caracteres provocarán una presentación irregular ya que el espacio libre sobreaparecerá en color amarillo. Una solución podría haber sido colocar en la instrucción DATA los espacios necesarios. Sin embargo, es un método tedioso en el que es fácil que se cometan errores.

Una solución más sencilla consiste en observar la longitud de la cadena más larga y la de la más corta. Añadiremos una cadena auxiliar (D$) al final de cada una de las cadenas con un número de espacios igual a la diferencia de longitudes entre ambas cadenas.

Hecho esto, nos desprenderemos, mediante instrucciones LEFT$, de un número de caracteres igual a la longitud de la cadena más larga. Esto parece muy complicado pero, en realidad, es muy sencillo (Figura 10.1). Recuerde que tendrá que sacar el RETURN del final de la línea 50000 cuando añada la línea 50010.

```
20 PRINT @ 111,USING"$#######.##+
";IN;:PRINT @ 143,USING"$#######.
##+";EX;:PRINT @ 175,USING"$#####
##.##+";BK;:PRINT @ 207,USING"$##
#####.##+";TI;:PRINT @ 239,USING
"$#######.##+";TW;:IF TW>TG THEN
30000 ELSE IF TW<CL THEN 20000 E
LSE IF BK<0 THEN GOSUB 41000
50010 DIM C$(6):DIM P$(6):D$=STR
ING$(6," "):FOR N=1 TO 6:READ C$
(N),P$(N):C$(N)=LEFT$(C$(N)+D$,1
1):PRINT @ (257+(32*N)),N;C$(N);
:NEXT N:RETURN
```

```
EN QUE COMPA/IA QUIERE INVERTIR
SU DINERO?
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~RENTA          ~~$    698.94+~~~~~~~
~GASTOS         ~~$   1129.57+~~~~~~~
~BALANCE BAN.   ~~$    458.83-~~~~~~~
~OTROS ACTIV.   ~~$   9700.00+~~~~~~~
~TOTAL ACTIV.   ~~$   9241.17+~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~ 1 CLAIRSIN     ~~~~~~~~~~~~~~~~~~
~ 2 ACHRON       ~7.1~6.8~$   300~
~ 3 COMPUTERS    ~3.8~0.4~$  2000~
~ 4 ATRIA VIDEO  ~1.1~8.5~$  4000~
~ 5 GRANDO       ~1.7~0.1~$  3000~
~ 6 TV PM        ~8.6~4.3~$   400~
~~~~~~~~~~~~~~~~~IN~~~EX~~~~~~~~~~

[ ~ REPRESENTA AQUI A CHR$(159)]
```

Figura 10.1 Presentación en pantalla de «Una cartera de valores más amplia»

Ahora que ya tenemos los nombres de las compañías y los números en pantalla, necesitamos colocar una línea que nos pregunte en qué compañía queremos invertir nuestro dinero. Para ello tendremos que entrar el número de la compañía que hemos escogido (C) (Diagrama de flujo 10.3). Después de esta línea se salta a las subrutinas apropiadas para cada compañía (ON C GOSUB). La función principal de estas subrutinas es dar nuevos valores a los porcentajes de rentas y de gastos de una compañía cada vez que resulta escogida. Estos porcentajes son distintos para cada compañía así que los almacenaremos separadamente en las matrices IR(C) y ER(C). Mientras dimensionamos estas matrices, podemos preparar otra que almacene el total invertido en cada compañía (TI(C)). Obsérvese que se ha añadido una comprobación que decide si se ha entrado un número de compañía correcto o no (línea 40).

```
30 PRINT @ 0,"EN QUE COMPA/IA":P
RINT @ 32,"QUIERE INVERTIR SU DI
NERO","";:INPUT C
40 IF C<1 OR C>6 THEN 30 ELSE ON
 C GOSUB 1000,2000,3000,4000,500
0,6000
1000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
2000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
3000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
4000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
5000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
6000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
50010 DIM C$(6):DIM P$(6):D$=STR
ING$(6," "):FOR N=1 TO 6:READ C$
(N),P$(N):C$(N)=LEFT$(C$(N)+D$,1
1):PRINT @ (257+(32*N)),N;C$(N);
:NEXT N:DIM IR(6):DIM ER(6):DIM
TI(6):RETURN
```

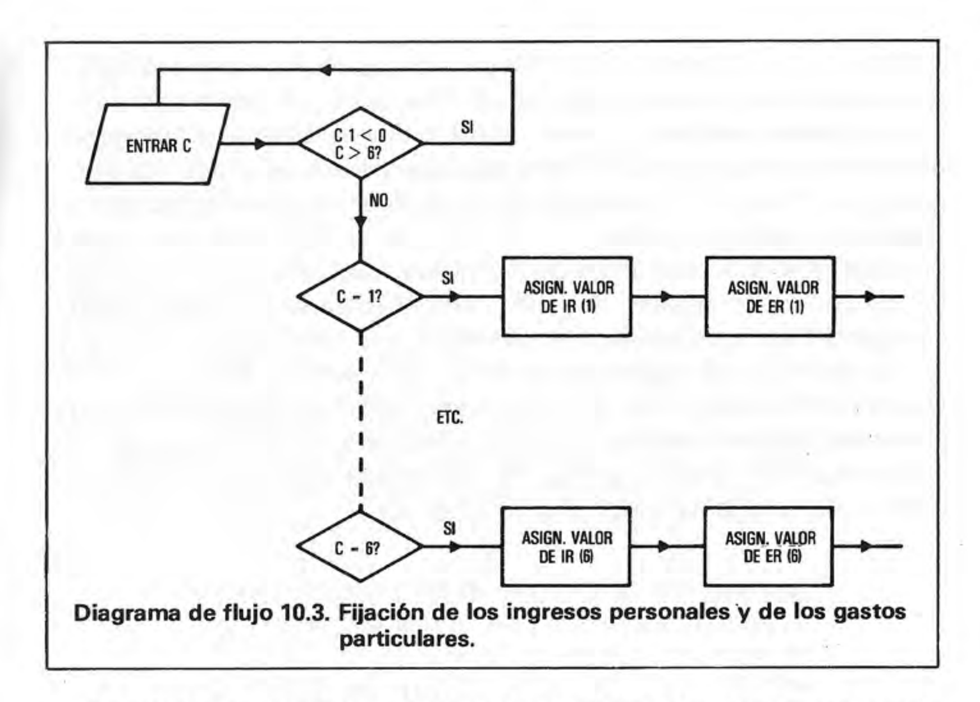


Diagrama de flujo 10.3. Fijación de los ingresos personales y de los gastos particulares.

Se pueden presentar en pantalla los últimos porcentajes de interés y gastos que se habían previsto para cada compañía (estimación basada en los resultados obtenidos en la anterior inversión). Debe recordarse, sin embargo, que estas cifras pueden haber variado y que, en cualquier caso, se trata tan sólo de límites extremos sujetos a las fuerzas del mercado representadas por RND(0).

```
40 IF C<1 OR C>6 THEN 30 ELSE ON
 C GOSUB 1000,2000,3000,4000,500
0,6000:PRINT @ (272+(C*32)),USIN
G "#.#";IR(C)*10:PRINT @ (278+(C
*32)),USING "#.#);ER(C)*10
50010 DIM C$(6):DIM P$(6):D$=STR
ING$(6," "):FOR N=1 TO 6:READ C$
(N),P$(N):C$(N)=LEFT$(C$(N)+D$,1
1):PRINT @ (257+(32*N)),N;C$(N);
:NEXT N:DIM IR(6):DIM ER(6):DIM
TI(6):PRINT @ 497,"IN";:PRINT @
503,"EX";:RETURN
```

Como siempre resulta útil saber qué es lo que fabrica nuestra compañía (C$(C)), lo recordaremos indicando el nombre del producto (P$(C)) que fabrica cuando preguntemos qué inversión se quiere hacer. La nueva inversión (IV(C)) se añade a la inversión total en esa compañía (TI(C)) y a la inversión total global TI. También podemos presentar en pantalla cuánto hemos invertido en cada compañía.

```
40 IF C<1 OR C>6 THEN 30 ELSE ON
 C GOSUB 1000,2000,3000,4000,500
0,6000:PRINT @ (272+(C*32)),USIN
G "#.#";IR(C)*10:PRINT @ (276+(C
*32)),USING "#.#);ER(C)*10;
50 PRINT @ 0,"CUANTO VA A INVERT
IR EN";C$(C);" ";P$(C);
60 PRINT @ 55,"";:INPUT IV(C)
70 IF BK-IV(C)<0 THEN 40000 ELSE
 BK=BK-IV(C)
80 TI=TI+IV(C):TI(C)=TI(C)+IV(C)
:PRINT @ 282+(C*32),USING"$#####
#";TI(C);
```

Si queremos tener en cuenta el dinero invertido en cada compañía y los correspondientes porcentajes de intereses y gastos en cada empresa, tiene que modificarse el cálculo de lo que nos rinde nuestra inversión.

Tenemos que dar el valor cero a los ingresos (IN) y a los gastos (EX) totales y luego realizar los cálculos necesarios, compañía por compañía y sumar, por un lado, los ingresos y, por otro, los gastos de cada una de ellas, para conseguir los valores totales que se procesan como antes.

```
90 IN=0:EX=0:FOR N=1 TO 6:IN=IN+
(IV(N)*(IR(N)*RND(0))):EX=EX+(IV
(N)*(IR(N)*RND(0))):NEXT N
```

LA OBTENCION DE MENSAJES DE UN TELEX SIMULADO

Para no ser menos que las grandes compañías financieras multinacionales nosotros también tendremos un télex que imprima men-

sajes. Entre los mensajes que nos informan del estado de nuestras finanzas y el listado de las compañías, podemos introducir nuestro mensaje de télex. Consistirá en imprimir el contenido de una cadena de forma que el texto vaya desplazándose lateralmente. Para conseguir este efecto haremos PRINT@ en una posición determinada mientras vamos partiendo la cadena carácter a carácter mediante MID$.

SUBRUTINA PARA LA IMPRESION DE MENSAJES

Para mejorar un poco el aspecto del mensaje (M$) haremos unas cuantas modificaciones. Entre ellas, habrá un encabezamiento amarillo realizado con STRING$(32,CHR$(159)) (de forma que empecemos en la parte derecha de la pantalla). También emplearemos indicadores de principio y fin del mensaje (CHR$(127)) y CHR$(159) (para borrar el final de la última parte de la cadena que se haya impreso) (Figura 10.2). Añadiremos CLEAR 1000 en la línea 10 para conseguir más espacio extra en el que almacenar nuestros mensajes (Diagrama de flujo 10.4).

```
10 CLEAR 1000:GOSUB 50000
45000 M$=STRING$(32,CHR$(159))+C
HR$(127)+M$+CHR$(127)+CHR$(159):
FOR ME=1 TO LEN(M$):PRINT @ 256,
MID$(M$,ME,32);:SOUND 1,1:NEXT M
E:RETURN
```

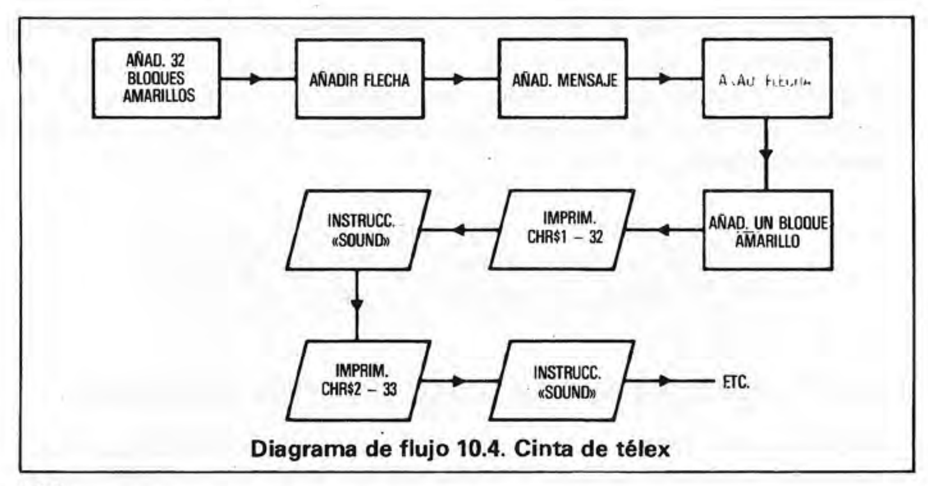


Diagrama de flujo 10.4. Cinta de télex

```
                               ←
                              ←H
                             ←HE
                            ←HER
                           ←HERE
                          ←HERE
                         ←HERE I
                        ←HERE IS
                       ←HERE IS
                      ←HERE IS A
                     ←HERE IS A
                    ←HERE IS A T
                   ←HERE IS A TI
                  ←HERE IS A TIC
                 ←HERE IS A TICK
                ←HERE IS A TICKE
               ←HERE IS A TICKER
              ←HERE IS A TICKERT
             ←HERE IS A TICKERTA
            ←HERE IS A TICKERTAP
           ←HERE IS A TICKERTAPE
          ←HERE IS A TICKERTAPE
         ←HERE IS A TICKERTAPE M
        ←HERE IS A TICKERTAPE ME
       ←HERE IS A TICKERTAPE MES
      ←HERE IS A TICKERTAPE MESS
     ←HERE IS A TICKERTAPE MESSA
    ←HERE IS A TICKERTAPE MESSAG
   ←HERE IS A TICKERTAPE MESSAGE
  ←HERE IS A TICKERTAPE MESSAGE←
 ←HERE IS A TICKERTAPE MESSAGE←
←HERE IS A TICKERTAPE MESSAGE←
HERE IS A TICKERTAPE MESSAGE←
ERE IS A TICKERTAPE MESSAGE←
RE IS A TICKERTAPE MESSAGE←
E IS A TICKERTAPE MESSAGE←
 IS A TICKERTAPE MESSAGE←
IS A TICKERTAPE MESSAGE←
S A TICKERTAPE MESSAGE←
 A TICKERTAPE MESSAGE←
A TICKERTAPE MESSAGE←
 TICKERTAPE MESSAGE←
TICKERTAPE MESSAGE←
ICKERTAPE MESSAGE←
CKERTAPE MESSAGE←
KERTAPE MESSAGE←
ERTAPE MESSAGE←
RTAPE MESSAGE←
TAPE MESSAGE←
APE MESSAGE←
PE MESSAGE←
E MESSAGE←
 MESSAGE←
MESSAGE←
ESSAGE←
SSAGE←
SAGE←
AGE←
GE←
E←
←
```

Figura 10.2. Reducción paulatina de la cadena para crear el efecto de cinta de télex.

Esta subrutina puede llamarse desde cualquier punto del programa con tal de que nos acordemos de definir, mediante M$, el mensaje que deseemos imprimir. Un ejemplo de utilización de la subrutina podría ser para realizar pronósticos referentes a las actividades de otras compañías. Para ello realizaríamos una selección (SE) de los mensajes adecuados, selección que podría estar relacionada con factores que tuvieran que ver con las ganancias y pérdidas de esas compañías.

```
1000 IR(C)=RND(0):ER(C)=RND(0):S
E=RND(5):ON SE GOTO 1010,1020,10
30 ELSE RETURN
1010 M$="CLIF CLAIRSIN NEGO ROTU
NDAMENTE LOS RUMORES DE QUE EL E
XPECTUM HUBIERA SIDO DISE/ADO PO
R EL FAMOSO MODISTO PIER DARDIN"
:GOSUB 45000:ER(C)=ER(C)*1.2:RET
URN
1020 M$="UN PORTAVOZ DE CLAIRSIN
 ACABA DE ANUNCIAR QUE LOS NUEVO
S MICRODRIVES ESTAN A PUNTO DE S
ALIR DE FABRICA":GOSUB 45000:ER(
C)=ER(C)*1.5:RETURN
1030 M$="LA CADENA DE ALMACENES
X DICE QUE EN 29 DIAS DISPONDRAN
 DE LA NUEVA VERSION DE 10 MB DE
L EXPECTUM":GOSUB 45000:IR(C)=IR
(C)*5:RETURN
```

OPERACIONES MERCANTILES EN TIEMPO REAL

Tal y como está el juego ahora nos deja un tiempo ilimitado para tomar nuestras decisiones financieras. Sin embargo, en la vida real el tiempo no espera a nadie. Por tanto, vamos a ver cómo nos las arreglamos para convertir este juego en otro de simulación en tiempo real en el cual las ganancias y las pérdidas se acumulen mientras el jugador aún está decidiendo qué es lo que va a hacer (Diagrama de flujo 10.5). Naturalmente tendremos que cambiar el INPUT corres-

pondiente a la decisión tomada por usted por un INKEY$ ya que la primera instrucción detiene la ejecución del programa hasta que se entre una respuesta. Como tan sólo queremos entrar un carácter, basta con que cambiemos la instrucción INPUT del final de la línea 30 por otra que comprueba el INKEY$. Para realizar la instrucción ON GOSUB necesitamos que C tenga un valor numérico. Por tanto, obtendremos primero (mediante la función VAL) el valor numérico de la cadena que hemos entrado con INKEY$. Hemos de comprobar si el número obtenido está entre el 1 y el 6 ya que sólo los números comprendidos entre esos dos serán correctos. De esta forma, cualquier valor de INKEY$ hará que la ejecución pase a otro punto del programa.

```
30 PRINT @ 0,"EN QUE COMPA/IA QU
IERE INVERTIR":PRINT @ 32,"SU DI
NERO","";:Q$=INKEY$:IF VAL(Q$)<1
 OR VAL(Q$)>6 THEN 90 ELSE C=VAL
(Q$)
```

Si se introduce esta modificación, las cifras continuarán cambiando y su posición se irá actualizando mientras el inversor aún estará pensando qué hacer. Todo continuará así hasta que se pulse una tecla numérica. En ese momento, el programa se parará y la instrucción INPUT aguardará a que se entre la cantidad que se desea invertir. Lógicamente, si no se pulsa ninguna tecla, parecerá que no ocurre nada puesto que no se habrá hecho ninguna inversión. El programa quedará en un bucle sobre esta línea. Sería conveniente añadir una instrucción que, en caso de que se haya escogido una determinada compañía C, se produzca un salto directo a la pregunta sobre la cantidad a invertir (si ya se ha escogido una compañía, C será mayor que 0).

```
30 IF C>0 THEN 50 ELSE PRINT @ 0
,"EN QUE COMPA/IA QUIERE INVERTI
R":PRINT @ 32,"SU DINERO","";:Q$
=INKEY$:IF VAL(Q$)<1 OR VAL(Q$)>
6 THEN 90 ELSE C=VAL(Q$)
```

Hemos de cambiar la instrucción INPUT de la línea 60 (que introduce la cantidad de dinero a invertir) por otra instrucción INKEY$.

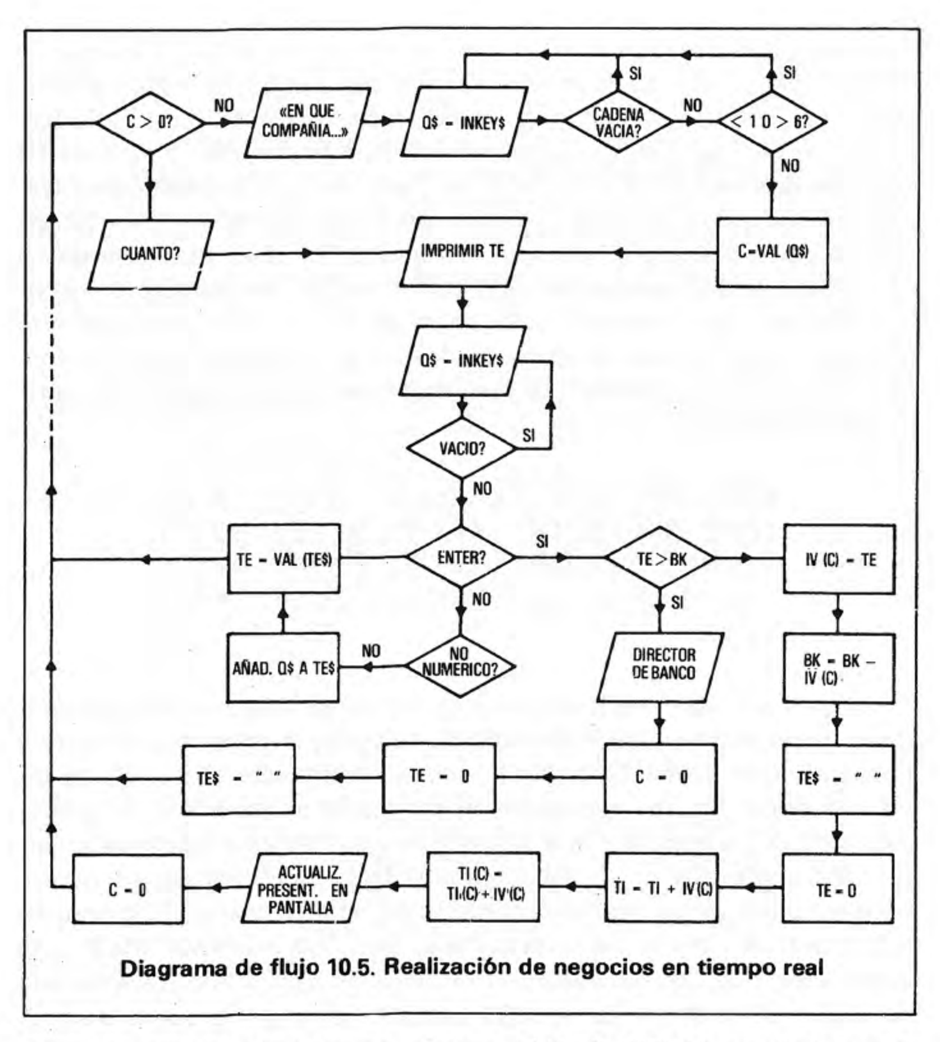


Diagrama de flujo 10.5. Realización de negocios en tiempo real

Esto presenta un problema y es que el VAL(INKEY$) nos limita a las cifras que van del 0 al 9. Para introducir inversiones mayores tendremos que entrar con INKEY$ una por una las cifras que forman la cantidad a invertir. Las iremos guardando en una cadena provisional (TE$) cuyo valor numérico será TE.

```
60 PRINT @ 55,"";:Q$=INKEY$:IF Q
$="" THEN 90 ELSE TE$=TE$+Q$:TE=
VAL(TE$):GOTO 90
```

La lista de los números que deseamos introducir se irá formando en TE$ y TE. Sin embargo, no podremos verlos a menos que pongamos una instrucción que los imprima (PRINT TE) (Recuérdese que la instrucción INPUT presenta automáticamente en pantalla los valores que se le entran).

```
60 PRINT @ 55,TE;:Q$=INKEY$:IF Q
$="" THEN 90 ELSE TE$=TE$+Q$:TE=
VAL(TE$):GOTO 90
```

Ahora estamos formando la cantidad de dinero que deseamos invertir pero no le hemos dado al programa ningún medio por el que nos pueda indicar si hemos terminado de entrar la cantidad total que deseamos invertir. Una solución lógica consiste en emplear la tecla ENTER para señalizar el fin de la entrada de este dato. Detectaremos que se ha pulsado esta tecla comprobando si aparece el valor ASCII que le corresponde: el 13. Téngase en cuenta que debemos rechazar las cadenas nulas antes de comprobar su valor ASCII. Si no lo hacemos obtendremos un error FC. También tendremos que prever el caso de que se pulse una tecla alfabética en vez de una numérica. De nuevo, conseguiremos discriminar de qué tecla se trata mediante la consulta de los valores ASCII.

```
60 PRINT @ 50,TE;:Q$=INKEY$:IF Q
$="" THEN 90 ELSE IF ASC(Q$)=13
THEN 70 ELSE IF ASC(Q$)<47 OR AS
C(Q$)>57 THEN 90 ELSE TE$=TE$+Q$
:TE=VAL(TE$):GOTO 90
```

Con tal de que no pulsemos la tecla RETURN , TE$ y TE irán creciendo de acuerdo con las teclas que hayamos ido pulsando. Sin embargo el programa seguirá actualizando la presentación en pantalla con las cifras anteriores de que disponía. La razón de esto reside en que la ejecución saltará las líneas 70 y 80 que son las que comprueban nuestra cuenta bancaria y se encargan de las nuevas inversiones. Cuando se pulsa RETURN, se pasa a la línea 70 que debe ser modificada para que transforme TE en IV(C) antes de volver a calcular nuestra posición. También tenemos que asignar a TE$ la cadena nula y a TE y C el valor cero antes de utilizarlos otra vez.

```
70 IF TE>BK THEN 40000 ELSE IV(C
)=TE:BK=BK-IV(C):TE$="":TE=0
80 TI=TI+IV(C):TI(C)=TI(C)+IV(C)
:PRINT @ (280+(C*32)),TI(C);:PRI
NT @ (280(C*32)),USING"$#######"
;TI(C);:C=0
40000 PRINT @ 0,"NO TIENE ESA CA
NTIDAD","EN SU CUENTA SE/OR":SCR
EEN 1,0:SOUND 10,10:C=0:TE=0:TE$
="":GOTO 30
```

AHORA ES SU TURNO

1) Cambie el escenario financiero en que se desarrolla el juego por el del mercado de publicación de software. Las decisiones que habrá que tomar serán sobre qué programas desarrollar.
2) No hemos preparado ningún medio que le permita vender sus inversiones. Ocurre ahora que en cuanto usted sufre un balance negativo está prácticamente condenado a la ruina. Sería conveniente introducir la posibilidad de vender nuestras inversiones además de comprarlas.
   Pistas:
   a) Si intenta entrar números negativos, la comprobación de los valores ASCII en la línea 60 los rechazará.
   b) Asegúrese de que comprueba si hay inversiones negativas.
3) Introduzca más factores variables que incidan sobre sus posibilidades de supervivencia. Piense acerca del empleo de la función SIN para determinar las tendencias del mercado a largo plazo.

Listado completo de «Negocios en tiempo real»

```
10 CLEAR 1000:GOSUB 50000
20 PRINT @ 111,USING"$######.##+
";IN;:PRINT @ 143,USING"$######.
##+";EX;:PRINT @ 175,USING "$###
###.##+";BK;:PRINT @ 207,USING"$
######.##+";TI;:PRINT @ 239,USIN
```

```
G"$######.##+";TW::IF TW>TG THEN
 30000 ELSE IF TW<CL THEN 20000
ELSE IF BK<0 THEN GOSUB 41
30 IF C>0 THEN 50 ELSE PRINT @ 0
,"EN QUE COMPA/IA QUIERE INVERTI
R":PRINT @ 32,"SU DINERO","";:Q$
=INKEY$:IF VAL(Q$)<1 OR VAL(Q$)>
6 THEN 90 ELSE C=VAL(Q$)
40 IF C<1 OR C>6 THEN 20 ELSEON
C GOSUB 1000,2000,3000,4000,5000
,6000:PRINT @ (272+(C*32)),USING
 "#.#";IR(C)*10;:PRINT @ (276+(C
*32)),USING"#.#";ER(C)*10;
50 PRINT @ 0,"CUANTO VA A INVERT
IR EN",C$(C);" ";P$(C)
60 PRINT @55,TE;:Q$=INKEY$:IF Q$
="" THEN 90 ELSE IF ASC(Q$)=13 T
HEN 70 ELSE IF ASC(Q$)<47 OR ASC
(Q$)>57 THEN 90 ELSE TE$=TE$+Q$:
TE=VAL(TE$):GOTO 90
70 IF TE>BK THEN 40000 ELSE IV(C
)=TE:BK=BK-IV(C):TE$="":TE=0
80 TI=TI+IV(C):TI(C)=TI(C)+IV(C)
:PRINT @ (280+(C*32)),TI(C);:PRI
NT @ (280+(C*32)),USING"$######"
;TI(C);:C=0
90 IN=0:EX=0:FOR N=1 TO 6:IF IV(
N)<1 THEN NEXT N ELSE  IN=IN+(IV
(N)*(IR(N)*RND(0))):EX=EX+(IV(N)
*(IR(N)*RND(0))):NEXT N
100 NI=IN-EX
110 BK=BK+NI
120 TW=BK+TI
200 GOTO 20
1000 IR(C)=RND(0):ER(C)=RND(0):S
E=RND(5):ON SE GOTO 1010,1020,10
30:RETURN
1010 M$="CLIF CLAIRSIN NEGO ROTU
NDAMENTE LOS RUMORES DE QUE EL E
XPECTUM HUBIERA SIDO DISE/ADO PO
R EL FAMOSO MODISTO PIER DARDIN"
```

```
:GOSUB 45000:ER(C)=ER(C)*1.2:GOT
O 50
1020 M$="UN PORTAVOZ DE CLAIRSIN
 ACABA DE ANUNCIAR QUE LOS NUEVO
S MICRODRIVES ESTAN A PUNTO DE S
ALIR DE FABRICA":GOSUB 45000:ER(
C)=ER(C)*1.5:RETURN
1030 M$="LA CADENA DE ALMACENES
X DICE QUE EN 29 DIAS DISPONDRAN
 DE LA NUEVA VERSION DE 10 MB DE
L EXPECTUM":GOSUB 45000:IR(C)=IR
(C)*5:RETURN
2000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
3000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
4000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
5000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
6000 IR(C)=RND(0):ER(C)=RND(0):R
ETURN
20000 CLS:PRINT @ 72,"ESTA ARRUI
NADO";:PRINT @ 136,"DEBE AL BANC
O";BK;:PRINT @ 260,"PERO SOLO DI
SPONE DE",TI:FOR N=255 TO 1 STEP
 -5:SOUND N,1:NEXT N
20010 PRINT @ 384,"SE ESCAPA A S
URAMERICA";:SOUND 250,5:PRINT @
450,"O QUIERE INTENTARLO OTRA VE
Z?";:SCREEN 0,1:Q$=INKEY$:IF Q$=
"" THEN 20010 ELSE RUN
30000 CLS:PRINT @ 233,"FELICIDAD
ES":PRINT @ 292,"CONSIGUIO SU OB
JETIVO EN";:PRINT @ 361,TIMER/50
;"SEGUNDOS";:FOR N=1 TO 255 STEP
 5:SOUND N,1:NEXT N:RUN
40000 PRINT @ 0,"NO TIENE ESA CA
NTIDAD","EN SU CUENTA SE/OR":SCR
EEN 0,1:SOUND 10,10:C=0:TE=0:TE$
="":GOTO 30
```

```
41000 FOR N=1 TO 10:PRINT @ 0,"S
U CUENTA ESTA EN DESCUBIERTO!!!"
,"TIENE QUE PAGAR UN INTERES DEL
 20%":SOUND 1,2:NEXT N:BK=BK-(AB
S(BK)*0.2):RETURN
45000 M$=STRING$(32,CHR$(159))+C
HR$(127)+M$+CHR$(127)+CHR$(159):
FOR ME=1 TO LEN(M$):PRINT @ 256,
MID$(M$,ME,32);:SOUND ME,1:NEXT
ME:RETURN
50000 CLS2:BK=10000:TIMER=0:TG=1
000000:CL=-1000000:PRINT @ 97,"R
ENTA";:PRINT @ 129,"GASTOS";:PRI
NT @ 161,"BALANCE BANCARIO";:PRI
NT @ 193,"OTROS ACTIVOS";:PRINT
@ 225,"TOTAL ACTIVOS";
50010 DIM C$(6):DIM P$(6):D$=STR
ING$(6," "):FOR N=1 TO 6:READ C$
(N),P$(N):C$(N)=LEFT$(C$(N)+D$,1
1):PRINT @ (257+(32*N)),N;C$(N);
:NEXT N:DIM ER(6):DIM TI(6):PRIN
T @ 497,"IN";:PRINT @ 503,"EX";:
RETURN
60000 DATA CLEARSIN,EXPECTUM,ACH
RON',ILLUSION,COMPUTERS,JYNX,AREK
 INT'L,AREK 1.6 K,GRANDO,DOGRAN
23,TV PM,MODELO T
```

Capítulo XI

LA ESPECTACULARIDAD EN LA PRESENTACION DE PROGRAMAS

Puesto que el lector ha llegado a este capítulo ya debe saber diseñar y desarrollar sus propios juegos de ordenador. Sin embargo, quizás le falte aún la apariencia profesional propia del software comercial de calidad. Vamos a echar una mirada en este último capítulo a la inclusión de los toques finales necesarios para alcanzar ese objetivo.

REUTILIZACION DE LA PRESENTACION DE UN PROGRAMA

Ha de tenerse en cuenta que la primera impresión es muy importante. La primera que recibe quien vaya a jugar al juego que se haya diseñado viene determinada por la secuencia de presentación de los títulos. Ya hemos dado detalles en otros programas anteriores acerca de cómo diseñar títulos espectaculares. Ahora estudiaremos una idea bastante diferente que puede darnos una presentación realmente impresionante. Si prepara un diseño gráfico de alta resolución en la pantalla, lo podrá guardar en un cassette como fichero en código máquina para, más tarde, cargarlo de nuevo desde el cassette y emplearlo como título de otro programa. Para guardar páginas de pantalla se emplea CSAVEM. Es necesario especificar los lugares de la memoria en los que empiezan y acaban las páginas así como calcular la diferencia entre ellos. Por ejemplo, para guardar las primeras cuatro páginas gráficas hay que entrar como una instrucción directa lo siguiente:

CSAVEM	"SCREEN"	1536,	7679,	6143,
		(inicio página 1)	(final página 4)	(diferencia)

(Para guardar otras páginas consúltese el mapa de memoria del Dragón y se podrán calcular las cifras apropiadas).

Para volver a cargar las páginas más tarde hacemos CLOADM. Si se quieren cargar en unas páginas distintas hay que especificar el desplazamiento sufrido por las direcciones de memoria. Por ejemplo, si se guarda un fichero con CSAVEM como antes y luego se hace CLOADM, 1535 cada página se habrá desplazado una página de forma que la página 1 será ahora la 2, etc.

Utilizando esta técnica se evita tener que reconstruir el dibujo cada vez que deseemos jugar. Basta con cargar de nuevo desde el cassette el dibujo que habíamos preparado con anterioridad. Por ejemplo, la figura 11.1 muestra un dibujo que podría resultar adecuado como título de una batalla espacial. Es conveniente dar instrucciones al jugador para que haga CLOADM y luego cargar el programa principal. También se puede poner CLOAD como una instrucción al principio del programa en BASIC.

```
10 CLS:PRINT @ 224,"PULSAR LA TE
CLA 'PLAY' DEL CASSETTE Y LUEGO
LA TECLA 'ENTER'":INPUT Q$:PMODE
3,1:SCREEN 1,0:CLOAD M
```

LA PRESENTACION DE UN PROGRAMA EN PANTALLA: INMEDIATA O DIFERIDA

No se olvide que, si se quiere ver el dibujo, habrá que especificar el PMODE y SCREEN. Tal y como está la línea, se podrá ver cómo se va desarrollando el dibujo sobre la pantalla a medida que va siendo transferido desde el cassette. Si se quiere que el dibujo permanezca oculto hasta que esté completamente terminado, basta con colocar la instrucción SCREEN al otro lado de CLOADM y añadir la línea provisional 20 (que se llama a sí misma). De esta forma se podrá ver el dibujo completo.

```
10 CLS:PRINT @ 224,"PULSAR LA TE
CLA 'PLAY' DEL CASSETTE Y LUEGO
LA TECLA 'ENTER'":INPUT Q$:PMODE
 3,1:CLOADM:SCREEN 1,0
20 GOTO 20
```

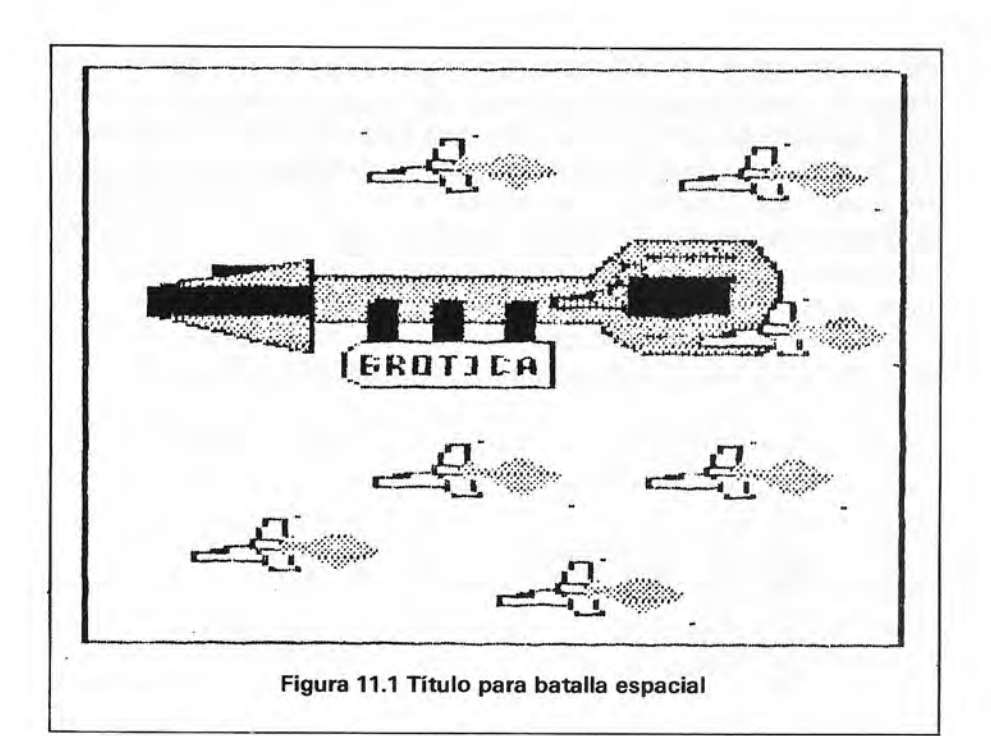


Figura 11.1 Título para batalla espacial

Vamos a pensar por un momento en cómo podemos hacer el dibujo original. La idea más sencilla consiste en escribir un programa que se encargue de dibujar nuestro diseño según unas instrucciones preestablecidas. Lógicamente esta solución requiere mucha planificación previa al tiempo que no permite mucha libertad artística.

Por otra parte, es posible preparar un programa «artístico» que utilice INKEY$ y nos permita acceder directamente a las instrucciones gráficas del Dragón al tiempo que nos deje ver la pantalla de alta resolución. Hay un inconveniente y es que el Dragón no puede poner textos en la pantalla de alta resolución. Esta desventaja puede superarse dibujando los caracteres. Al final del capítulo damos los detalles de un programa que permite alcanzar ambos objetivos simultáneamente.

LA INCLUSION DE INSTRUCCIONES CLARAS EN LOS JUEGOS

No hace falta decirlo: las personas que vayan a jugar a su mag-

nífico juego serán mentalmente inferiores al autor del mismo. Obviamente, necesitarán instrucciones claras acerca de cómo jugar. Estas tendrán que aparecer al principio aunque resulta conveniente que puedan ser consultadas más adelante cuando, durante el desarrollo del juego, aparezcan dudas acerca de algún punto específico. La forma más sencilla de conseguir esto es distribuyendo, en puntos estratégicos del juego, instrucciones que comprueben si se ha entrado una «i» (en cuyo caso es que se quieren consultar las instrucciones). Un lugar adecuado para colocar estas comprobaciones es junto a cada petición de información que se haga al jugador.

```
60 INPUT X$:IF X$="I" THEN GOSUB
 1000 ELSE .............
or
60 X$=INKEY$:IF X$="" THEN 60 EL
SE IF X$="I" THEN GOSUB 1000
or
60 INPUT X:IF ASC(X)=73 THEN GOS
UB 1000 ELSE .............
```

COMO EVITAR LA DESTRUCCION DE UN PROGRAMA POR ERROR

Aunque, sin duda, se ha preparado un conjunto de excelentes y muy claras instrucciones, el jugador puede entrar voluntaria o involuntariamente un dato incorrecto. El resultado de este error puede ser despreciable o catastrófico. En ambos casos, el jugador siempre echará la culpa al programa. Ya hemos tratado anteriormente acerca de cómo enfrentarnos con alguna de estas situaciones pero vamos a recordar algunos puntos en concreto.

Si se intenta entrar una letra cuando se pida una variable numérica (X por ejemplo), aparecerá el mensaje de error ?REDO que puede causar un desastre con la presentación en pantalla que con tanto cuidado habíamos planificado. No se puede evitar este mensaje de error. La mejor solución consiste en entrar los datos como cadenas, mediante la instrucción INPUT y luego convertir éstas en una variable numérica (en caso de que se necesite un número). Recuérdese que la función VAL da como resultado el valor numérico de una cadena

pero es siempre 0 cuando se aplica a letras. Si pueden entrarse tanto letras como números, conviene comprobar todas las teclas válidas. Al final se debe añadir un bucle que retorne al principio de la petición de datos de entrada cuando se haya pulsado una tecla incorrecta.

CREACION DE NIVELES DE DIFICULTAD EN UN JUEGO

Los buenos juegos siempre resultan difíciles de dominar completamente. Es una buena idea preparar distintos niveles de habilidad para que el jugador pueda ir progresando a través de ellos de manera que, al llegar al más alto, ya haya adquirido una buena preparación. La forma exacta en que se preparen estos niveles depende de la estructura del juego pero, normalmente, se recurre al cambio de unas determinadas variables. Si pensamos en «A toda marcha», por ejemplo, nos daremos cuenta de que cuanto más ancha es la carretera, más fácil es permanecer en ella. Podríamos establecer los niveles de habilidad cambiando el 10 de STRING$ de la línea 20 de forma que se incluya una nueva variable SK. Después añadiríamos una petición de información al jugador para que nos indicara en qué nivel quiere jugar. En el nivel 2 A$ está formado como antes por 12 − 2 = 10 bloques. En el nivel 1, el más fácil, hay 12 − 1 = 11 bloques y al nivel más avanzado hay 12 − 3 = 9 bloques. Obsérvese que se han preparado mecanismos de alta seguridad como los que acabamos de comentar.

```
20 PRINT "NIVEL DE HABILIDAD",,,
"1=APRENDIZ",,"2=NORMAL",,"3=CON
DUCTOR EXPERTO";:INPUT SK:IF SK<
1 OR SK>3 THEN 20 ELSE A$=STRING
$(12-SK,128):FOR N=1 TO 16:PRINT
TAB(10);A$:NEXT N:TIMER=0
```

En los programas espaciales se puede modificar fácilmente el nivel de dificultad cambiando las posibilidades de que un vehículo extraterrestre dispare contra nosotros o bien alterando el ángulo o la distancia a los que puede disparar nuestro fáser. El programa «Aterri-

zaje» es aún más fácil de modificar puesto que el factor crítico aquí es la anchura del pasadizo que hay que atravesar para llegar a la plataforma de aterrizaje. Si se observa de nuevo la figura en la que se mostraba la presentación en pantalla de este programa, se podrá comprobar que todo lo que se tiene que hacer es cambiar los dos puntos inferiores de la línea que indicaba el límite superior del pasadizo.

RUTINA PARA UNA TABLA DE GANADORES

Las tablas de ganadores son un componente esencial de los juegos por ordenador. Vamos a preparar una rutina que se encargue de confeccionar una tabla de este tipo y que pueda ser empleada en cualquier programa. Aunque simplemente presenta en pantalla una lista de nombres y puntuaciones tenemos que considerar dos cuestiones más para darle un aire realmente profesional (Diagrama de flujo 11.1 y figura 11.2).

1) Cada nueva puntuación debe ponerse en la tabla únicamente si es superior a alguna de las anteriores.
2) En las tablas de ganadores las cifras siempre aparecen con ceros por la izquierda.

```
TABLA DE PUNTUACIONES
---------------------
99900000  PACO
84567000  SEBASTIAN
70068500  NICOLAS
60430040  ISABEL
51002000  CATALINA
40002001  MIGUEL
30004500  ROBERTO
21003000  JOAQUIN
10000001  MARIA
00000001  ALFREDO
---------------------
```

Figura 11.2. Tabla de puntuaciones

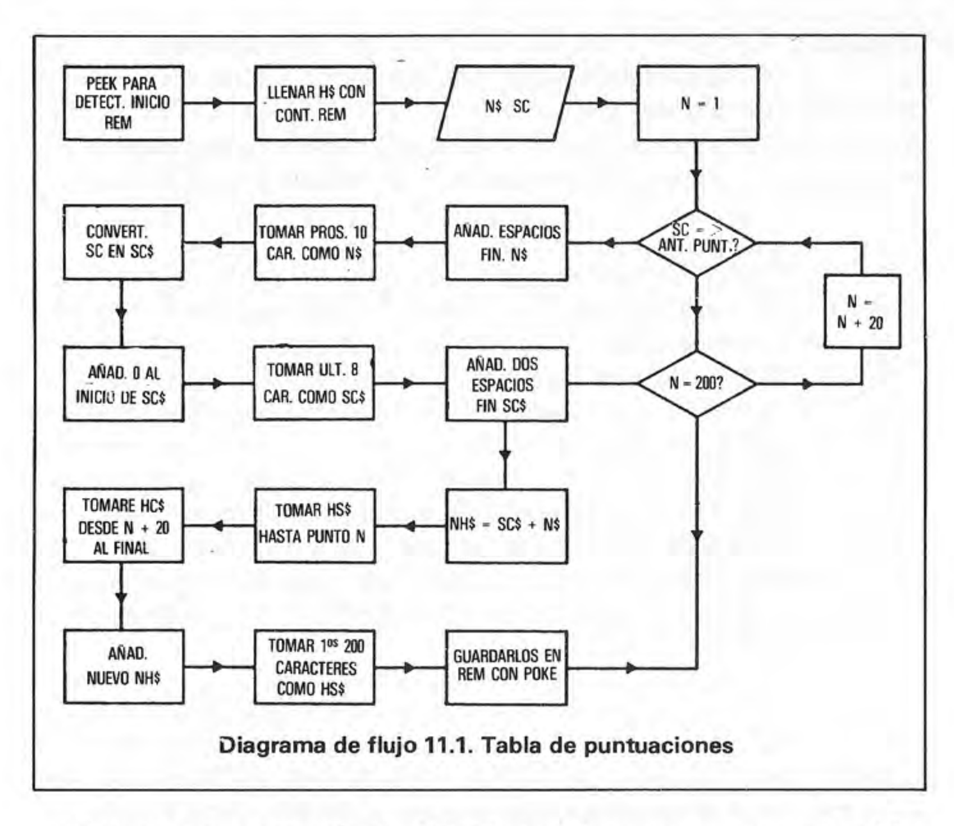


Diagrama de flujo 11.1. Tabla de puntuaciones

Tenemos que empezar decidiendo qué formato va a tener la tabla. Vamos a fijarlo en ocho números (para la puntuación) / dos espacios / diez letras (para el nombre). Si queremos tener un máximo de 10 nombres en la tabla necesitaremos un total de 200 caracteres. Ahora bien, la longitud máxima de una cadena es de 255 caracteres: toda la tabla nos cabría en una sola cadena (HS$). Para poner inicialmente en marcha el sistema guardaremos en HS$ algunas cifras ficticias (línea 10). Luego, en la línea 20, pediremos que el jugador entre su nombre (N$). La puntuación que se haya conseguido (SC) será pedida en la línea 30. (En un programa real de juego la puntuación se introduce automáticamente para impedir que el jugador haga trampa).

Hay que comprobar ahora si la puntuación obtenida por el jugador supera a la mínima puntuación guardada en HS$. Para esto tenemos que dividir HS$ en subcadenas de 20 caracteres cada una

(mediante FOR N = 1 TO 200 STEP 20) y luego comparar el valor numérico de los primeros ocho caracteres de la subcadena con la puntuación del jugador mediante IF SC = > VAL (MID$(HS$,N,8))). Si la puntuación no es mayor o igual que alguna de las que ya hay en la cadena, se retrocede a la petición de nombre y puntuación.

```
10 H$="90000000    PACO    80000
000  SEBASTIAN 70000000   NICOLA
S  60000000   ISABEL   50000000
 CATALINA  40000000   MIGUEL   3
0000000  ROBERTO   20000000  JOA
QUIN    10000000   MARIA    00000
001  ALFREDO    "
20 PRINT "TU NOMBRE?";:INPUT N$
30 PRINT "CUANTOS PUNTOS HAS CON
SEGUIDO";:INPUT SC
40 FOR N=1 TO 200 STEP 20:IF SC=
>VAL(MID$(HS$,N,8)) THEN 50:ELSE
 NEXT N:GOTO 20
```

Por otra parte, si la puntuación es mayor o igual que una de las puntuaciones que ya están en la tabla, tenemos que colocarla en la posición correcta dentro de la tabla (y de la cadena). El primer paso consiste en tomar el nombre del jugador y añadirle unos cuantos espacios en blanco para que su longitud sea de 10 caracteres (Línea 50). Luego, se toman los primeros 10 caracteres de la cadena como el nombre del jugador (si su nombre tiene más de 10 caracteres, mala suerte pero, al menos, de esta manera no puede usted alterar el formato de la tabla) (línea 60).

Para asegurarse de que la puntuación siempre tiene ceros a la izquierda, se añaden ocho ceros al principio de SC$ a los que siguen las cifras que componen la puntuación obtenida (línea 70). Recuérdese que STR$(SC) siempre nos da un espacio inicial que debe eliminarse haciendo MID$ a partir del segundo carácter. Luego se toman los ocho últimos caracteres de SC$ (línea 80), se añaden dos espacios (línea 90), la puntuación y el nombre juntos con lo que se obtiene una nueva cadena de puntuaciones (NH$) (línea 100).

```
50 N$=N$+STRING$(10," ")
```

```
60 N$=LEFT$(N$,10)
70 SC$=STRING$(8,"0")+MID$(STR
$(SC),2)
80 SC$=RIGHT$(SC$,7)
90 SC$=SC$+"   "
100 NH$=SC$+N$
```

La nueva puntuación ha de colocarse en el lugar adecuado de la cadena donde se guarda la tabla de puntuaciones. Esto resulta fácil ya que disponemos de un puntero al lugar adecuado (N). (Recuérdese que N guardaba el punto que habíamos alcanzado antes de pasar a la comparación con los distintos bloques de HS$). Tomamos la parte izquierda de HS$ hasta N, la añadimos a la nueva cadena (NHS$) y luego agregamos lo que sea necesario de HS$ para conseguir nuevamente 200 caracteres (línea 110). Finalmente, la línea 120 imprime la cadena en bloques de 20 caracteres cada uno colocados uno encima de otro. La línea 130 salta de nuevo a la línea de petición de nombre y puntuación.

```
110 HS$=LEFT$(HS$,N)+NH$+MID$(HS
$,N,181-N)
120 FOR N=1 TO 200 STEP 20:PRINT
 MID$(HS$,N,20):NEXT N
130 GOTO20
```

FICHERO PARA CONSERVAR LAS TABLAS DE GANADORES

La única desventaja de este programa es que, en cuanto desconectemos el Dragón, perderemos el contenido de la tabla y siempre empezará el juego con los mismos nombres. Es perfectamente posible conservar el contenido de HS$ abriendo un fichero de datos que deberá guardarse y cargarse en o desde el cassette independientemente del programa. Pero este método es tedioso. Una solución alternativa consiste en colocar los caracteres en el programa mediante instrucciones POKE. Así HS$ se guardará o almacenará con el propio programa cuando lo transfiramos al cassette.

El secreto está en hacer que la última línea del programa sea un REM asegurándonos de que se rellene con los caracteres suficientes

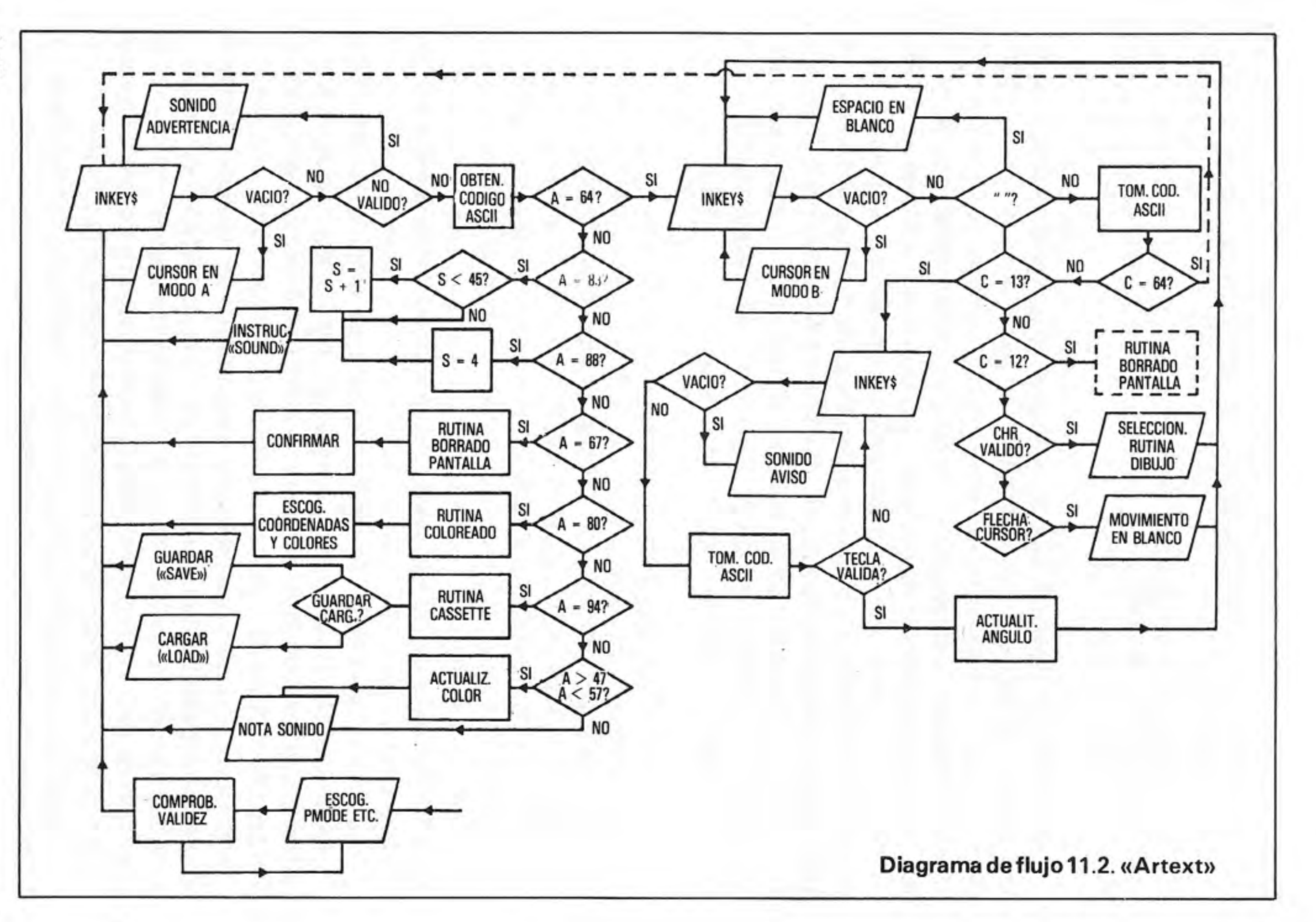


Diagrama de flujo 11.2. «Artext»

para guardar toda la información que tenemos que conservar. El sistema ignora cualquier cosa que venga detrás de un REM. Esto permite almacenar en esa línea cualquier información que deseemos. Si, como hemos aconsejado, se hace esto en la última línea del programa, se podrá aplicar la instrucción PEEK a las posiciones 27 y 28 que señalan al fin del programa. Así se sabrá en qué lugar de la memoria se encuentra esa información. Si se coloca el contenido de la cadena HS$ original en la línea 63999 (borrando la línea 10 al mismo tiempo) y luego se rellena completamente (por ejemplo, con el carácter "."), el PEEK (27)* 256 + PEEK (28) − 251 (= PS) apuntará a la letra M de la palabra REM que hay en 63999. De esta forma, podremos reconstruir HS$ cada vez que deseemos jugar. Para ello sacaremos los caracteres de la línea 63999 mediante CHR$ (PEEK(PS + N)). Cuando se quiera guardar la nueva tabla de puntuaciones que se haya obtenido tras una partida se podrá transferir cada uno de los caracteres de HS$ al espacio que hay detrás de REM. Para ello, se empleará la instrucción POKE que colocará los valores ASCII en PS + N (Diagrama de flujo 11.2).

```
63900 PS=PEEK(27)*256+PEEK(28)-2
51
63910 FOR N=1 TO 201:HS$=HS$+CHR
$(PEEK(PS+N)):NEXT N
63920 FOR N=1 TO LEN(HS$):POKE P
S+N,ASC(MID$(HS$,N,1)):NEXT N
63999 REM90000000    PACO    800
00000  SEBASTIAN 70000000  NICOL
AS   60000000    ISABEL    500000
00  CATALINA  40000000    MIGUEL
  30000000  ROBERTO   20000000
JOAQUIN   10000000    MARIA   00
000001   ALFREDO................
................................
```

Con tanto PEEK y POKE se consume bastante tiempo. Es aconsejable llamar a estas rutinas específicas únicamente al principio y al final del programa. Hagamos que la primera línea del programa llame a la subrutina de la línea 63900 encargada de cargar HS$. Añadamos luego una instrucción CSAVE al final de la línea que carga

el REM (63920). De esta forma podremos guardar fácilmente el programa y la tabla de puntuación conjuntamente haciendo GOTO 63920.

```
1 GOSUB 63900
63900 PS=PEEK(27)*256+PEEK(28)-2
51
63910 FOR N=1 TO 201:HS$=HS$+CHR
$(PEEK(PS+N)):NEXT N:RETURN
63920 FOR N=1 TO LEN(HS$):POKE P
S+N;ASC(MID$(HS$,N,1)):NEXT N:CSA
VE "NOMBRE"
```

Obsérvese que en el listado completo que se da más adelante:

1) Se ha sumado 63000 a todas las líneas de numeración inferior de forma que no interfieran con el programa que se desarrolle.
2) La rutina tiene que llamarse como una subrutina mediante GOSUB 63020.
3) Se ha borrado la petición de puntuación mediante INPUT así que hay que asegurarse antes de saltar a esta subrutina de que asigna a la puntuación el valor de SC.

EL EFECTO DE PERSPECTIVA EN LOS GRAFICOS

Aunque ya hemos hablado de muchos aspectos del empleo de los gráficos, no hay suficiente espacio en este libro para cubrir todos los detalles de las posibilidades gráficas del Dragón. Se podrían escribir varios libros tan sólo refiriéndose a los gráficos de alta resolución. Sin embargo, dada la importancia de los gráficos en la calidad de un juego, vamos a dar unos cuantos ejemplos e ideas más que requieren gráficos bastante más complejos que los utilizados hasta ahora.

Hemos explicado el empleo de laberintos tridimensionales pero no hemos dicho cómo se puede proyectar una imagen en tres dimensiones. De hecho, resulta muy fácil. Para conseguir un efecto lo más real posible hay que trabajar en el modo de mayor resolución (el 4) que sólo tiene el color blanco y el negro. El efecto tridimensional se

obtiene dibujando una serie de rectángulos uno dentro de otro y uniendo luego sus esquinas. (Figura 11.3).

```
10 PMODE 4,1:PCLS:BX=125:SCREEN
1,0
125 REM COORDENADAS MAXIMAS
130 X1=8:X2=244:Y1=8:Y2=180
135 REM DIBUJAR SERIE DE RECTANG
ULOS
140 FOR N1=1 TO BX STEP 25
150 LINE(X1+N1,Y1+N2)-(X2-N1,Y2-
N2),PSET,B
160 N2=N2+15:NEXT N1:BX=BX-25:N2
=N2-15
165 REM CONECTAR ESQUINAS
170 LINE(X1,Y1)-(X1+BX,Y1+N2),PS
ET
180 LINE(X1,Y2)-(X1+BX,Y2-N2),PS
ET
190 LINE(X2,Y1)-(X2-BX,Y1+N2),PS
ET
200 LINE(X2,Y2)-(X2-BX,Y2-N2),PS
ET
215 REM PRESENTAR FIGURA
220 PMODE 4,1:SCREEN 1,0
225 REM ESPERAR A PROXIMO VALOR
BX
230 B$=INKEY$:IF B$="" THEN 230
ELSE BX=VAL(B$)*25:PCLS:N2=0:GOT
O 130
```

El número de rectángulos dibujados depende del valor de BX y, en este listado de demostración, la línea 230 permite cambiar ese número pulsando la tecla numérica apropiada. Esta se utiliza para calcular el nuevo valor de BX. Para usar esta técnica en sus propios programas ha de relacionarse esta variable con la distancia que hay hasta el siguiente pasadizo. Se consigue un efecto aún más real si se modifica la figura para que no sea totalmente simétrica.

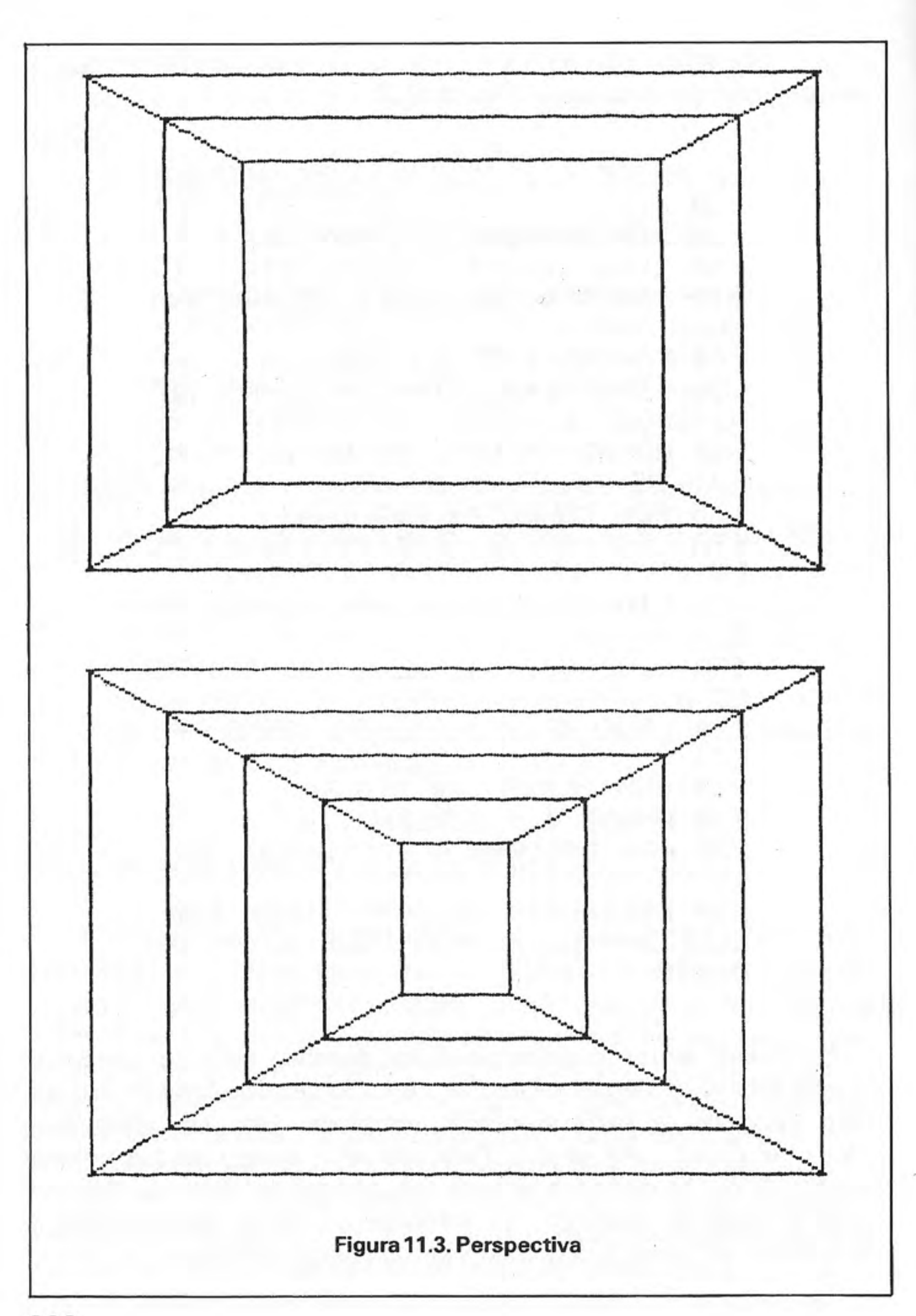


Figura 11.3. Perspectiva

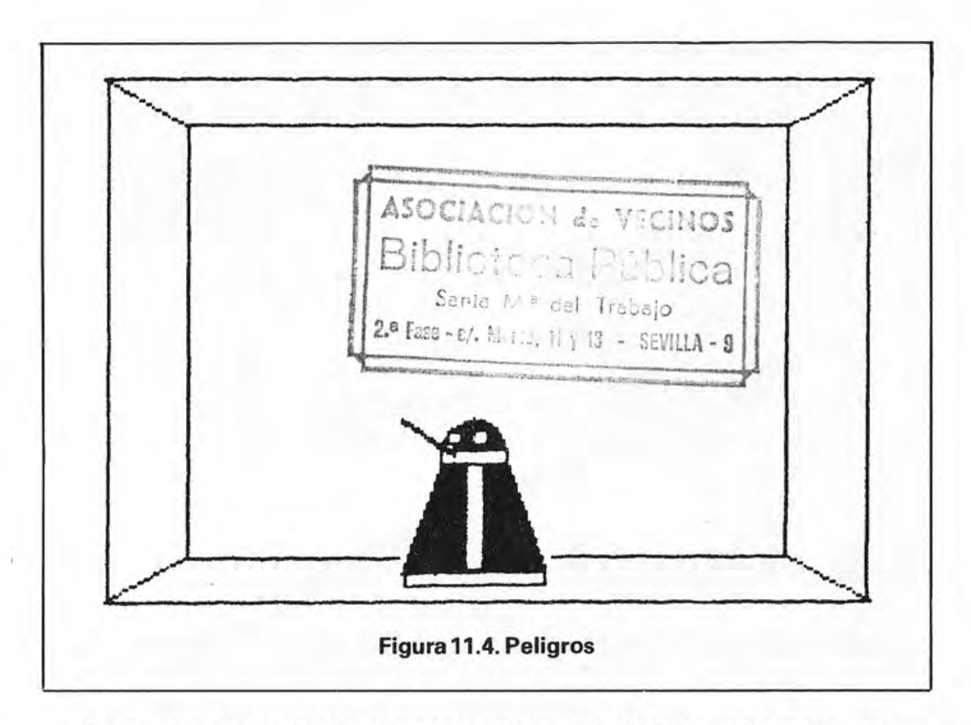


Figura 11.4. Peligros

Desde luego no hay que sorprenderse si súbitamente se ve aparecer peligros en su camino (Figura 11.4).

La figura descrita en las siguientes líneas se construye combinando varias instrucciones DRAW, LINE, CIRCLE y PAINT y luego aplicando GET para almacenarla en una matriz. Luego, con una instrucción PUT, se la puede hacer aparecer en los lugares y momentos apropiados.

```
15 REM DIMENSIONAR LA MATRIZ
20 DIM A1(0,100)
25 REM CONSTRUIR FIGURA
30 DRAW"BM100,100EUUEUUEUUEUUEUU
EUUEUUEUUEUUEUUEUUEUUR20FDDFDDFD
DFDDFDDFDDFDDFDDFDDFDDFDDFDDL43D
4R43U4
40 LINE(112,64)-(132,59),PSET,B
50 CIRCLE(122,59),10,1,1,0.5
60 LINE(120,64)-(125,99),PSET,B
```

```
70 LINE(116,61)-(100,50),PSET:LI
NE(116,60)-(100,49),PSET
80 CIRCLE(116,55),3:CIRCLE(124,5
5),3
90 PAINT(115,80),1,1:PAINT(135,8
0),1,1
100 PAINT(123,52),1,1
105 REM HACER GET CON LA FIGURA
110 GET(100,45)-(145,105),A1,G
115 REM BORRAR FIGURA
120 PCLS
205 REM VOLVER A CREAR LA FIGURA
210 PUT(100,115)-(145,175),A1,PS
ET
```

LA REALIZACION DE MAPAS EN LA PANTALLA

Resulta conveniente contar con un mapa de vez en cuando (Figura 11.5). Una forma sencilla de dibujarlo consiste en utilizar el programa «artístico» Artext. Con él podemos crear el mapa sobre la pantalla de alta resolución y transferirlo de nuevo al programa desde el cassette mediante CLOADM. Si se tiene vocación de mártir se puede intentar el empleo de PSET con cada punto que se haya guardado previamente en instrucciones DATA. Uno puede preguntarse por qué no relacionamos cada elemento de la pantalla con otro de una matriz. La respuesta es muy sencilla. Hay 48K elementos individuales en los modos PMODE 3 y 4. Es un poco difícil asignar un elemento de la matriz a uno de la pantalla. Si se utilizase el modo que emplea menos colores (PMODE 1) entonces esta cantidad se reduciría a 12 K que sigue siendo una cifra excesiva.

Sin duda, lo mejor que se puede hacer es intentar emplear la presentación en pantalla para controlar las consecuencias. Se puede disponer de cuatro colores y, por tanto, de cuatro posibilidades básicas que se pueden complicar mediante factores aleatorios o teniendo en cuenta nuestras coordenadas X e Y. Para el agua el color que, lógicamente, tenemos que emplear es el azul; para la hierba el verde; para los árboles, el amarillo (¡estamos en otoño!) y para las carreteras el rojo (son pistas forestales). Si se cae en el agua quizás

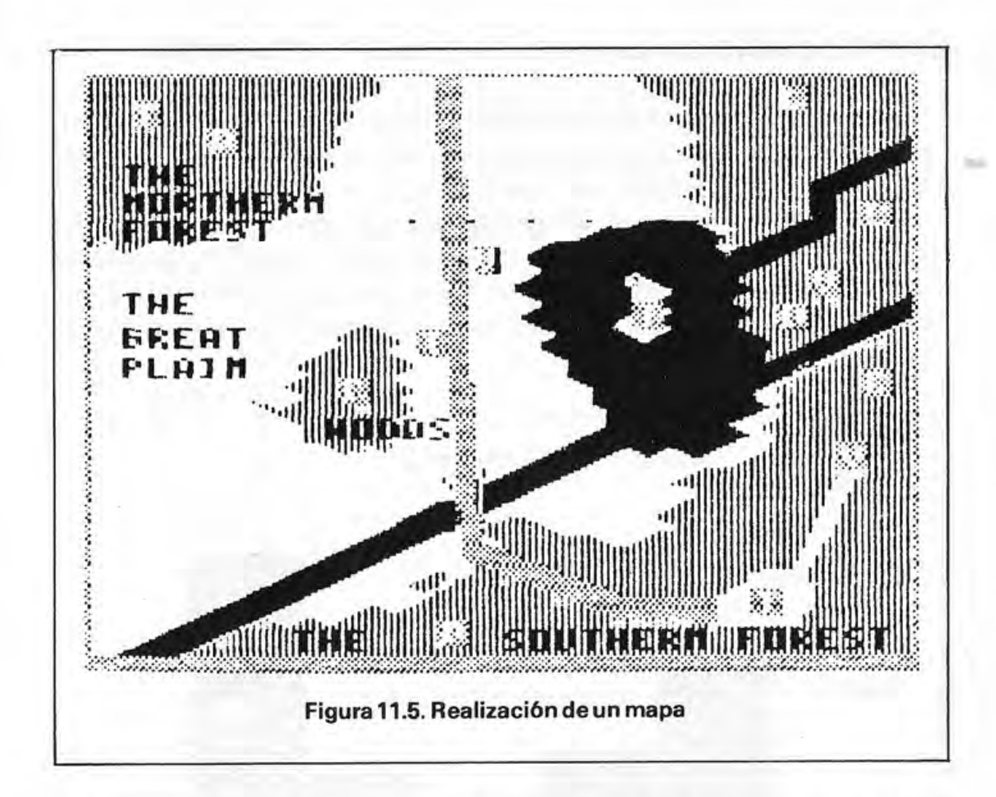


Figura 11.5. Realización de un mapa

no se pueda nadar y ¿quién sabe qué peligros se esconden en el bosque? Los lugares de interés, como las ciudades, cuevas o las guaridas de dragones se pueden dibujar en cualquier color que contraste con el fondo. Estos lugares se pueden detectar más fácilmente a partir de sus coordenadas que por su color.

Por descontado, habrá que poder controlar los movimientos y ver hacia dónde se dirige uno. Habrá que incluir una rutina de comprobación de teclas. Podrá moverse aplicando PSET a distintos puntos. Para pararse, dejando una estela tras de sí, habrá que llevar a cabo la siguiente secuencia de acciones: primero, ver qué hay en la siguiente posición a la que nos dirigimos mediante PPOINT; luego, almacenar provisionalmente el valor así obtenido; convertir el color del nuevo punto a nuestro color mediante PSET y luego hacer PSET sobre el anterior punto con el valor que habíamos almacenado provisionalmente.

LA PRESENTACION DE DATOS MEDIANTE GRAFICOS

En nuestro programa de finanzas presentábamos mediante números la información acerca de nuestra situación económica. Sería más interesante poder reflejar esa información con gráficos (Figuras 11.6 y 11.7) aunque esto no significará que se tenga que dibujar los textos mediante DRAW en vez de imprimirlos normalmente. La colocación de textos en los ejes es un buen ejemplo que muestra la utilidad que tiene el poder imprimir perpendicularmente a la dirección normal.

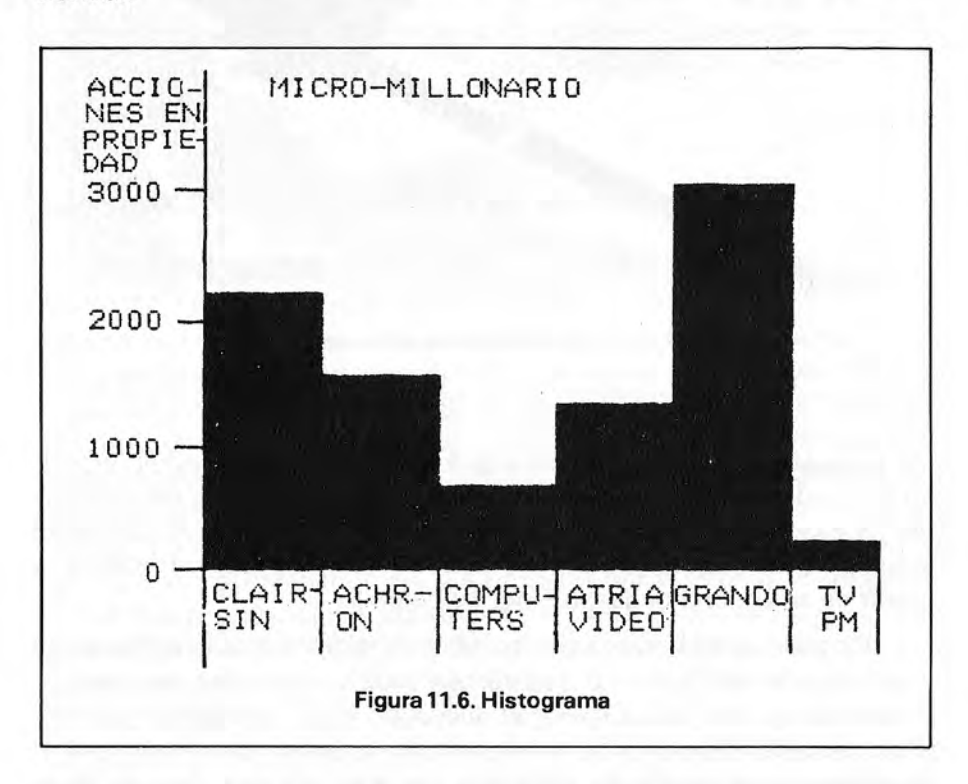


Figura 11.6. Histograma

TRUCOS PARA MEJORAR RUTINAS GRAFICAS

Volvamos atrás y echemos una mirada a la rutina gráfica que preparamos para el programa «A toda marcha». Veamos cómo la podemos mejorar con un poco de reflexión y unos cuantos truquillos.

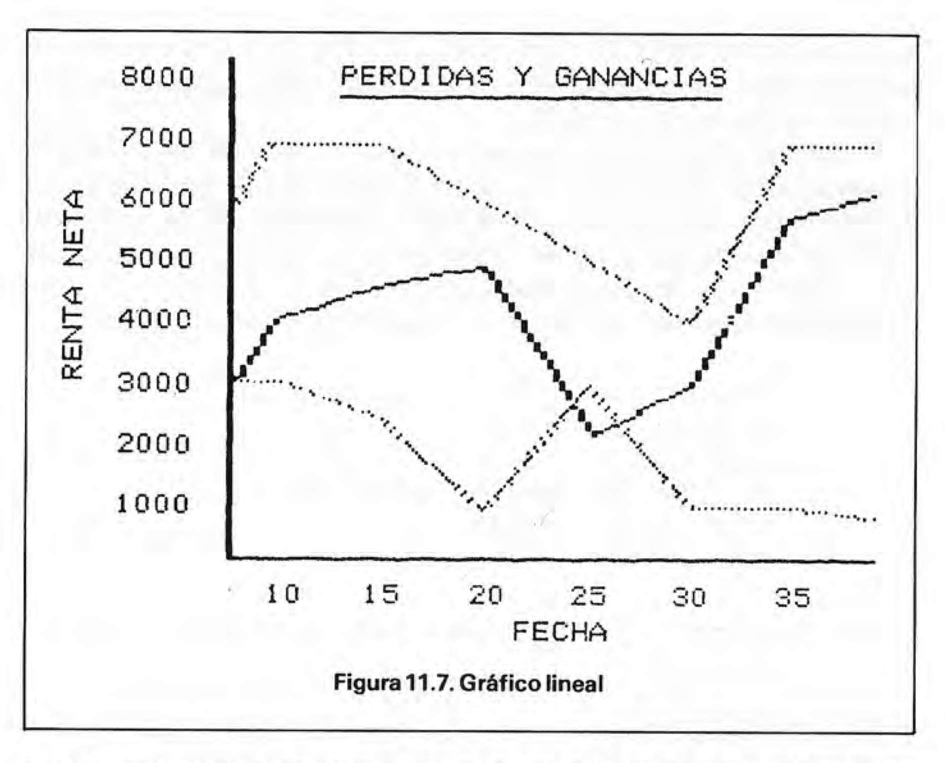


Figura 11.7. Gráfico lineal

Lo primero que debe quedar grabado en la cabeza es cómo utilizan las páginas gráficas los diferentes modo gráficos. El modo PMODE 1 requiere dos páginas y el PMODE 4 cuatro, pero podemos empezar cualquier modo en cualquier página. Más aún, podemos copiar cualquier página en cualquier otra mediante PCOPY sin necesidad de cambiar, al mismo tiempo, toda la plantilla. Si se sigue este ejemplo, paso a paso, se podrá ver cómo se resuelve una situación realmente compleja.

En primer lugar, tenemos que reservar las ocho páginas gráficas añadiendo PCLEAR 8 a la línea 850. Luego hay que cambiar el modo normal por PMODE 3,1 (de forma que podamos ver qué está pasando en las cuatro páginas) y, finalmente, debemos añadir un salto a la nueva subrutina que empieza en la línea 890. Durante la primera parte de esta modificación empleamos PMODE 1 y empezamos en la página 1; es decir, estamos trabajando con las páginas 1 y 2 a las que damos el color amarillo. Luego trazamos un dibujo muy sencillo que representa el espejo retrovisor por un rectángulo que habremos relle-

nado mediante LINE,BF y una sola LINE. Para comprobar cómo se ve todo esto, introdúzcanse las líneas 850 y 890, póngase la línea provisional 891 y hágase RUN.

Obsérvese que el retrovisor está en la parte superior de la pantalla. Es decir, está realmente en la página 1 (Figura 11.8). Ahora sáquese la instrucción SCREEN de la línea 891 y hágase RUN de nuevo. Se verá que sólo se utiliza la mitad superior de la pantalla (ya que el último SCREEN estaba en la línea 850 después de PMODE 3.1) y que la altura del retrovisor se ha reducido a la mitad (Figura 11.9).

```
850 PCLEAR 8:PMODE 3,1:SCREEN 1,
0:PCLS:GOTO 890
890 PMODE 1,1:PCLS 2:LINE (80,40
)-(140,80),PRESET,BF:LINE (110,4
0)-(110,0),PRESET
891 SCREEN 1,0:GOTO 891
```

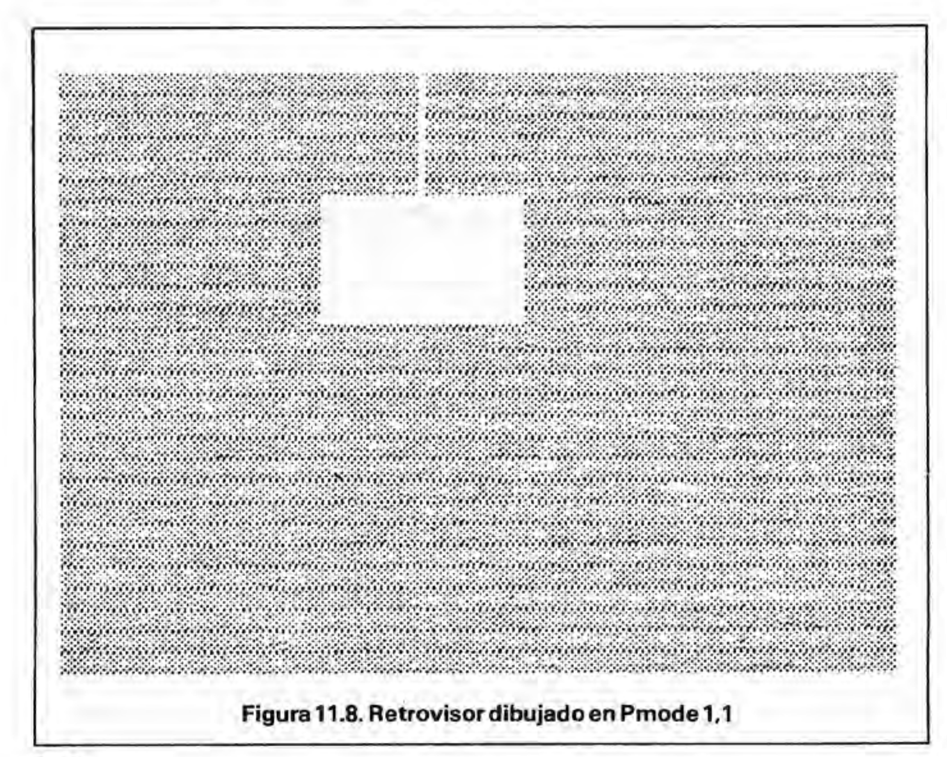


Figura 11.8. Retrovisor dibujado en Pmode 1,1

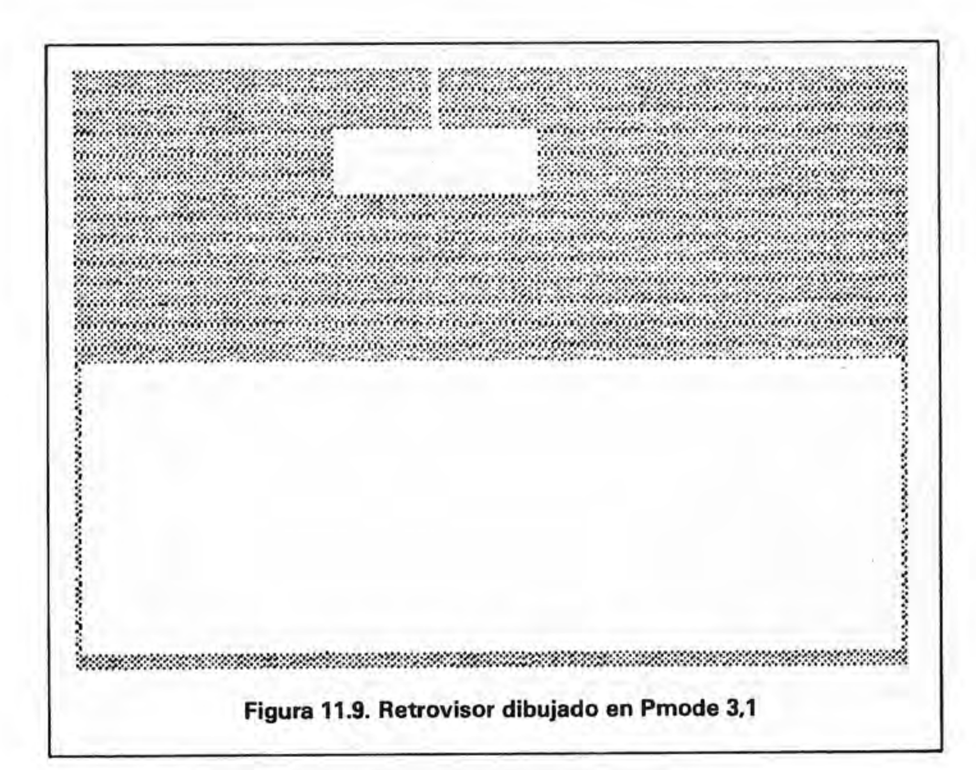


Figura 11.9. Retrovisor dibujado en Pmode 3,1

Ahora cambiamos la línea 855 de forma que tengamos PMODE en vez de PMODE 1,3 y pongamos LINE B para crear un rectángulo que ocupe toda la anchura de la pantalla. Cambiemos las páginas que copiamos en la línea 865 poniendo PCOPY 6 TO 3 y PCOPY 7 TO 3 en vez de PCOPY 3 TO 1 y PCOPY 4 TO 2. Cambiamos también el PMODE de la línea 880 por PMODE 3,1 y, finalmente, la línea 891 por GOTO 855 y hagamos RUN.

```
855 PMODE 3,5:FOR N=2 TO 21 STEP
 2:LINE(0,0)-(255,191),PSET,B
865 PCOPY 6 TO 2:PCOPY 7 TO 3
890 PMODE 3,1:FOR N=1 TO 255 STE
P 10:PLAY "T"+STR$(255-N)+"O1C05
BO3C":CIRCLE(100,140),N,2,0.5:NE
XT N
891 GOTO 855
```

CREACION DE UNA VENTANA EN EL CENTRO DE LA PANTALLA

Se comprobará que la página 1 se queda como estaba, de manera que el espejo ocupa la misma posición. Se podrá ver también que la página 2 queda sobreimpresa por la mitad superior del camión. Como las páginas 6 y 7 sólo son copias sobre las páginas 2 y 3 únicamente aparece la parte central de ese dibujo. Esto significa que se puede ver aún el espejo de la parte superior (página 1) y que la parte inferior de la pantalla (página 4) continúa siendo de color verde. Hemos creado, por tanto, una verdadera ventana en el centro de la pantalla a través de la cual estamos mirando (Figura 11.10). Mediante unas mínimas modificaciones a los parámetros de la línea 880 podemos mejorar las cosas un poco más. El centro del círculo utilizado para representar la rotura del parabrisas se traslada a la unión de las páginas 3 y 4; el número de círculos se reduce a 200 (para que no alcancen el techo del coche [página 1]) y ahora sólo los dibujamos en un arco de círculo que va desde la posición que, en un reloj, corresponde a las 9 en punto hasta la que corresponde a las 3 en punto (así los círculos no se solapan sobre la página 4).

```
880 PMODE 3,1:FOR N=1TO200 STEP
10:PLAY "T"+STR$(255-N)+"O1CO5BO
3C":CIRCLE(100,145),N,2,0.5,0.5,
0:NEXT N
```

DISEÑO DEL VOLANTE Y DEL SALPICADERO DE UN CAMION

Ahora prepararemos la presentación en pantalla correspondiente al volante y a los instrumentos del salpicadero. Lo haremos en la página 4. Dibujaremos en PMODE 1 en amarillo (COLOR 2) y el fondo será azul (COLOR 3) (Figura 11.11). Como la figura experimentará una distorsión cuando más tarde la pongamos en modo PMODE 3, tenemos que dibujar los círculos como elipses verticales distorsionadas. El volante se realiza dibujando primero dos arcos y haciendo luego PAINT entre ellos y añadiendo un radio del volante. Los distintos indicadores se crean mediante instrucciones CIRCLE. Las agujas correspondientes se añaden posteriormente con la instrucción LINE.

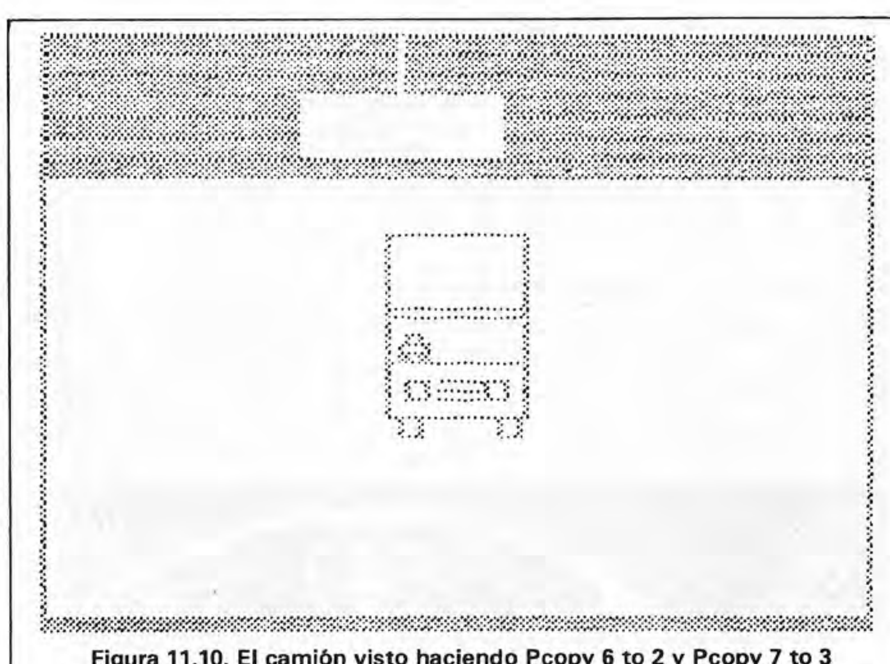


Figura 11.10. El camión visto haciendo Pcopy 6 to 2 y Pcopy 7 to 3

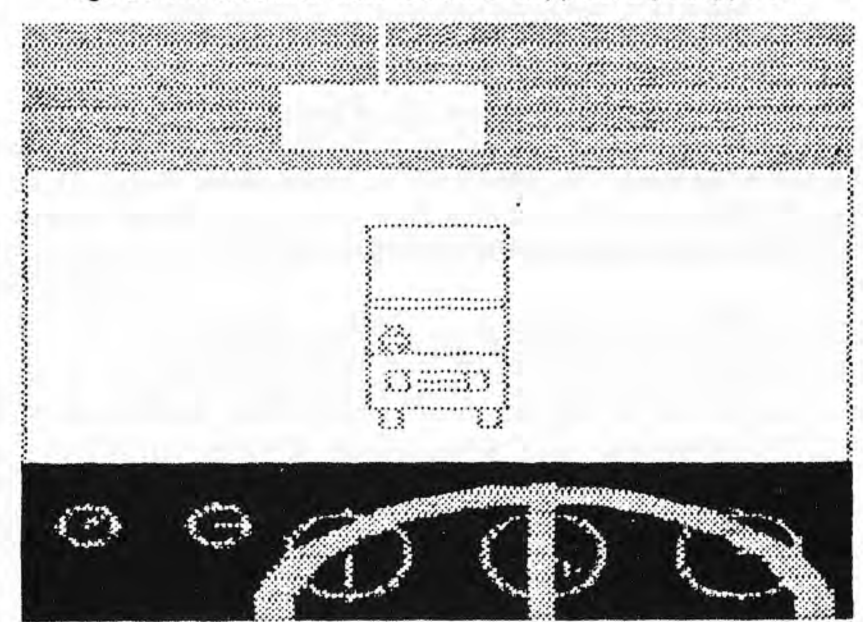


Figura 11.11. Volante e indicadores (Dibujados en Pmode 1 y vistos en Pmode 3)

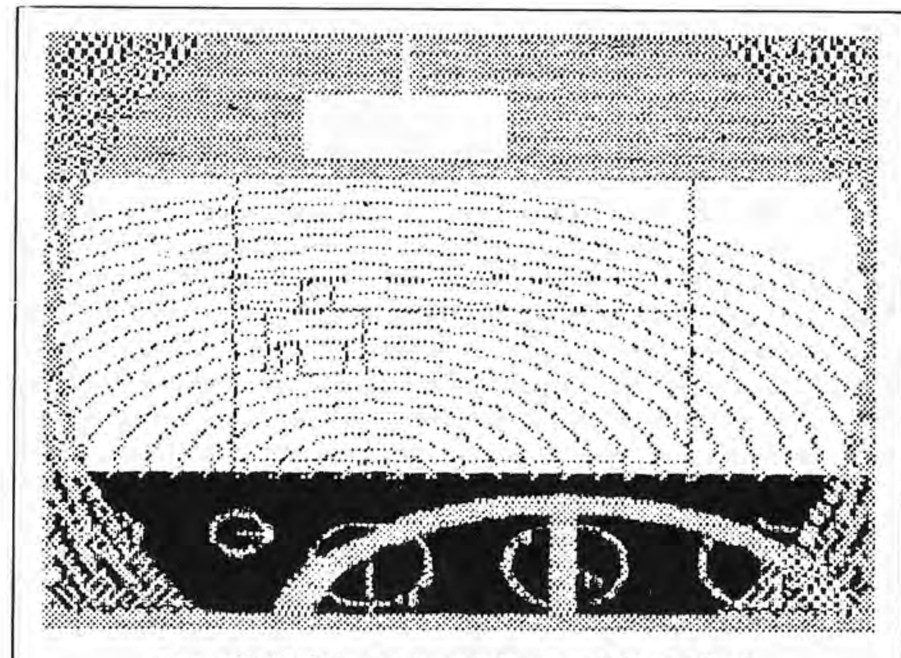


Figura 11.12. Rotura del parabrisas en el choque final

Como toque final, cuando el conductor pierde la consciencia cambiará el color de todo lo que se ve. Lo convertiremos en rojo de forma que sobreimpresione claramente los haces azules (Figura 11.12) Se han utilizado números impares para numerar las líneas anteriores a la 900 que es utilizada por diferentes rutinas.

```
895 FOR N=60 TO 1 STEP-1:PLAY"V"
+STR$(INT(N/2))+"01T255CDCDCDCD"
:CIRCLE(128,96),(N*3),4:NEXT N:R
UN
895 PMODE 1,4:PCLS3:COLOR 2,3:CI
RCLE(160,105),90,2:CIRCLE(160,10
5),90,2:PAINT(75,105),2,2:LINE(1
57,105)-(163,25),PSET,BF
896 CIRCLE (100,60),20,2,1.5:LIN
E(100,60)-(100,90),PSET:CIRCLE (
160,60),20,2,1.5:LINE(160,60)-(1
```

```
70,70),PSET:CIRCLE(220,60),20,2,
1.5:LINE(220,60)-(225,55),PSET:C
IRCLE(20,40),10,2,1.5:LINE(20,40
)-(25,35),PSET:CIRCLE (60,40),10
,2,1.5
897 LINE(60,40)-(70,40),PSET:GOT
O 855
```

Hasta el momento nos ha convenido no alterar la instrucción SCREEN de la línea 850. Así podíamos ver cómo la presentación en pantalla, iba apareciendo gradualmente. En realidad, sería mejor que apareciera de golpe. Si añadimos PCOPY 3 TO 2 al final de la línea que dibuja al retrovisor la página 2 cambiará a verde de forma inmediata. Si no utilizamos una instrucción SCREEN hasta que se haya dibujado todo nuestro vehículo, la presentación en pantalla aparecerá de forma instantánea después del corto tiempo que se necesita para que se ejecuten todas las instrucciones.

```
850 PCLEAR 8:PMODE 3,1:PCLS:GOTO
 890
890 PMODE 1,1:PCLS 2:LINE (90,40
)-(140,90),PRESET,BF:LINE (110,4
0)-(110,0),PRESET:PCOPY 3 TO 2
897 LINE(60,40)-(70,40),PSET:PMO
DE 3,1:SCREEN 1,0:GOTO 855
```

INCLUSION DE TEXTOS EN LA PANTALLA DE ALTA RESOLUCION

Aunque el sistema del Dragón no puede imprimir directamente textos en la pantalla de alta resolución, resulta fácil preparar una rutina que lo haga. Básicamente hay dos formas de colocar un texto en la pantalla. La primera consiste en aplicar la instrucción PSET a distintos puntos para formar el texto. Esta forma es muy lenta y requiere una gran cantidad de datos. La segunda forma emplea la instrucción DRAW para crear las letras. Una vez se han preparado las cadenas correspondientes cada letra es fácil dibujarlas a diferentes escalas con distintos colores y en ángulos diversos mediante la instrucción DRAW.

El diseño de caracteres específicos constituye en sí mismo un tema que está debidamente tratado en el libro «The Working Dragon» de David Lawrence. Nosotros no añadiremos más. Tan sólo nos limitaremos a explicar el programa correspondiente que ofrece todo lo que se pueda necesitar en los programas de juego. También se ha incluido una rutina que permitirá acceder a algunas de las instrucciones gráficas del Dragon a través de INKEY$ para que, así, se pueda dibujar directamente. (Diagrama de flujo 11.2).

Para emplear el programa, tecléese RUN y respóndase a las preguntas iniciales sobre la resolución que se desee, número de colores que se requieren, y color del fondo y del primer plano. Surgirá la pantalla de alta resolución con el color que se ha escogido para el fondo. Un pequeño cursor intermitente aparecerá en la esquina superior izquierda de la pantalla. Nos encontraremos ahora en modo «artístico» y, sobre el fondo que se ha escogido, se puede dibujar una línea vertical, horizontal o diagonal. Para ello hay que pulsar las teclas L (izquierda), R (derecha), U (arriba), D (abajo), E, F, G ó H. Para aumentar la longitud de la línea dibujada en cada movimiento (la escala) púlsese S. Para volver a la escala mínima, púlsese X. Para cambiar el color con el que se dibuja, púlsese la tecla correspondiente al número del nuevo color con el que desee dibujar. Para borrar basta con dibujar sobre las líneas ya trazadas sobre el color del fondo.
Para hacer PCLS púlsese la tecla C cuando se pida la confirmación de la decisión del usuario. Para colorear mediante PAINT una determinada zona púlsese P cuando se pidan las coordenadas, el color y el color del borde de la zona a colorear. Para pasar a modo texto, púlsese @ . Si, en ese momento, se pulsa una tecla alfanumérica, aparecerá el carácter @. La barra de espaciado produce un movimiento hacia la derecha en el que se imprime un espacio en blanco.
Las teclas del cursor permiten moverse por toda la pantalla. Si se quiere fijar el ángulo en el que se está dibujando, ha de pulsarse ENTER. Se oirá un sonido de advertencia que se interrumpirá cuando se haya pulsado una tecla situada entre el 0 y el 3 (0 = normal, 2 = hacia abajo, 3 = hacia arriba). Para volver al modo artístico ha de pulsarse @. Los cambios de escala y de color sólo se pueden hacer en modo artístico pero tienen efecto en modo texto.
Para almacenar un dibujo en el cassette o para transferirlo del casset-

te ha de pulsarse una flecha del cursor mientras se encuentre en modo artístico. Esto llevará a la rutina del cassette.

En este programa se emplean buena parte de las teclas con símbolos del teclado del Dragón. Hemos dejado algunos espacios en blanco para que el lector pueda colocar en ellos las teclas que desee asociar a otras operaciones que pueda diseñar. Obsérvese que el número de las líneas en las que hay una cadena para un carácter en particular utilizada por una instrucción DRAW es el mismo que para el código ASCII de ese carácter.

Además de emplear este programa para dibujar la secuencia de títulos de presentación para los programas de juego, las secciones de dibujo de textos pueden utilizarse en otros programas de alta resolución que utilicen textos y lleven cuenta de puntuaciones.

MEZCLA DE TEXTOS Y GRAFICOS EN LA PANTALLA

Como despedida vamos a mezclar textos y gráficos en la pantalla. Trabajaremos con el juego del «Aterrizaje» y de esta forma, lo actualizaremos (Figura 11.13). Como las rutinas de formación de textos ya están en el programa «Artext», lo más lógico será unir los dos programas en uno y luego realizar las modificaciones oportunas. En el Dragon es posible unir programas mediante APPEND pero resulta un tanto fastidioso. Lo primero que hay que hacer es CLOAD con el programa «Artext» y, luego, borrar las partes que no sean necesarias. Esto significa que hay que hacer DEL-24 y DEL-91. Nos quedará la línea 25 que salta a las líneas de dibujo de caracteres (33 a 90) según cuál sea el código ASCII de C$. Es necesario hacer PEEK sobre algunas posiciones reservadas de memoria y tomar nota de los valores que allí se encuentren. Introdúzcase la siguiente línea como una instrucción directa:

```
PRINT PEEK(25);PEEK(26);PEEK(27);PEEK(28)
```

Las dos primeras posiciones contienen un puntero al inicio del programa en BASIC. Las dos últimas, un puntero al final del programa. Si se han reservado cuatro páginas gráficas (la condición por defecto) las dos primeras posiciones contendrán los valores 30 y 1. Los valores que guarden las dos últimas posiciones dependerán de la lon-

gitud del programa. Si se han borrado las líneas que hemos indicado, los valores contenidos en esas posiciones deberán ser 37 y 201. Ahora se ha de engañar al sistema y hacerle creer que su programa no existe. Lo conseguirá introduciendo en las posiciones 25 y 26 nuevos valores mediante instrucciones POKE. Hay que hacer POKE 25 con el valor que estaba en la posición 27 y POKE 26 con el valor que había en la 28 menos dos unidades. Por tanto, introduzca las siguientes instrucciones directas:

```
POKE 25,37
POKE 26,199
```

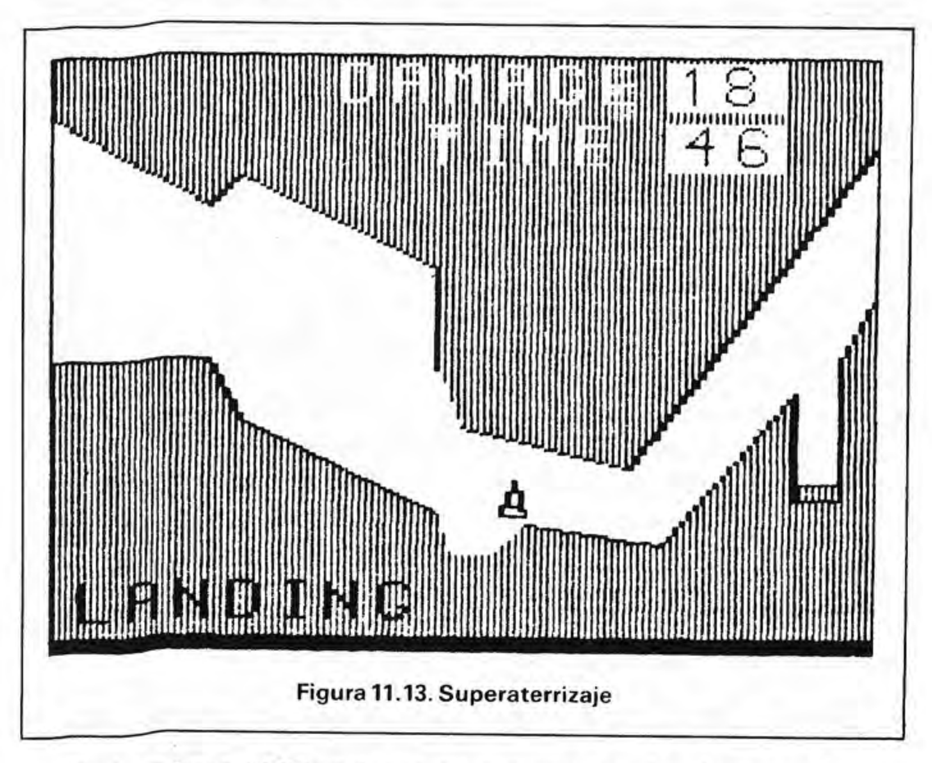


Figura 11.13. Superaterrizaje

Si ahora hace LIST verá que su programa se ha desvanecido. No se preocupe: se trata de una desaparición transitoria. Puede cargar con CLOAD el programa de aterrizaje como haría normalmente. Si hace LIST, todo aparecerá como de costumbre y si hace RUN el programa funcionará como siempre. Antes de unir los dos progra-

mas, es necesario volver a numerar las líneas del nuevo programa mediante RENUM para que todos los números de línea sean mayores que los utilizados en el anterior programa. El número de línea más alto de «Artext» que nos queda es el 90. Haremos RENUM 100 de forma que los números de «Aterrizaje» estén entre el 100 y el 260. Y ahora llega el momento de la verdad: volvemos a darle el valor anterior al puntero de inicio del programa en BASIC.

```
POKE 25,30
POKE 26,1
```

Si usted hace LIST ahora: ¡Maravillas! ¡Aleluya!: ambos programas son ahora uno solo.

Es el momento de hacer una o dos modificaciones. Antes que nada si queremos jugar tenemos que saltar por encima de los restos de «Artext». Por tanto, añadiremos la línea 10 que salta directamente a la 100.

```
10 GOTO 100
```

Hay un par de cosas al principio de la línea 25 que no son necesarias. Haremos EDIT y borraremos hasta el punto en que está ON... Como hemos eliminado el espacio original (Código ASCII 32) en «Artext» (ya que también actuaba como rutina de borrado), necesitamos cambiar ahora el número que deducíamos del código ASCII. Lo cambiaremos por 31, añadiremos 32 al principio de la lista de números y escribiremos una nueva línea 32 que sólo imprima un espacio en blanco a la derecha.

```
25 ON(ASC(C$)-31)GOSUB32,33,34,3
5,36,37,38,39,40,41,42,43,44,45,
46,47,48,49,50,51,52,53,54,55,56
,57,58,59,60,61,62,63,64,65,66,6
7,68,69,70,71,72,73,74,75,76,77,
78,79,80,81,82,83,84,85,86,87,88
,89,90:RETURN
32 DRAW"BM+8,+0":RETURN
```

Lo siguiente que haremos será considerar cómo vamos a obtener valores adecuados de C$ a partir de las palabras y letras que desee-

mos presentar en pantalla. Las tendremos en WD$ y, por desconta-do, necesitaremos ir dividiendo esta cadena, carácter a carácter. Por esta razón, añadiremos una rutina general en la línea 20 que se encargue de todo esto.

```
20 FOR N=1 TO LEN(WD$):C$=MID$(W
D$,N,1):GOSUB 25:NEXT N:RETURN
```

Una vez hechas estas modificaciones, podemos empezar a pensar dónde queremos que aparezca texto en la pantalla y cuándo tenemos que dibujarlo. Lógicamente, lo primero que hay que crear es el título. Puede dibujarse inmediatamente después que haya sido creado el paisaje en el que se desarrollará el juego. Dejaremos el título en la parte inferior izquierda de la pantalla, lejos de nuestro alcance. Para esto realizaremos un movimiento en blanco hacia la posición adecuada. Obsérvese que hemos empleado letras de doble tamaño pues hemos especificado el número 88 en vez del que el sistema toma por defecto (84). Obsérvese también que la impresión se realiza mediante GOSUB 20. El color empleado será el que el sistema toma por defecto para el primer plano en PMODE 3,1:SCREEN 1,1. Las palabras «daños» y «tiempo» se colocan en la esquina superior izquierda por un procedimiento similar. También se ha inicializado el temporizador.

```
175 WD$="ATERRIZAJE":DRAW"BM10,1
85"+"S8":GOSUB 20:WD$="DA/OS":DR
AW "C1BM90,15":GOSUB 20:WD$="TIE
MPO":DRAW"BM116,35":GOSUB 20:TIM
ER=0
```

Tenemos que preparar ahora una rutina que actualice el temporizador. También hemos de hacer que se imprima continuamente el valor del temporizador (línea 235). Para mayor claridad, se crea una zona en blanco aplicando la instrucción PUT sobre una matriz (A3) llena de ceros, y dimensionada en la línea 110. Así, se borra el tiempo que aparecía anteriormente mientras ya se está colocando el nuevo. El tiempo se expresa en segundos haciendo INT(TIMER/50). La línea 300 actualiza los daños sufridos por la nave al chocar. Como la actualización del texto disminuye un poco la velocidad de la ejecución la puede usted aumentar haciendo POKE&HFFDL7,0

en la línea 100 pero sin olvidar el hacer otra vez POKE antes de guardar o almacenar el programa en el cassette.

```
100 PMODE 3,1:SCREEN 1,1:PCLS:J0
=10:J1=70:T0=J0:T1=J1:POKE&HFFD7
,0
110 DIM A1(0,9):DIM A2(0,3):DIM
A3(0,20)
235 PUT(190,22)-(225,37),A3,PSET
:WD$=STR$(INT(TIMER/50)):DRAW"BM
190,35":GOSUB 20
300 PUT(190,2)-(225,17),A3,PSET:
WD$=STR$(DA):DRAW"BM190,15":GOSU
B 20:RETURN
```

Listado completo de la tabla de puntuaciones

```
1 GOSUB 63900
63020 PRINT "SU NOMBRE?";:INPUT;
N$
63040 FOR N=1 TO 200 STEP 20:IF
SC=>VAL(MID$(HS$,N,8)) THEN 6305
0:ELSE NEXT N:GOTO 63020
63050 N$=N$+STRING$(10,"  ")
63060 N$=LEFT$(N$,10)
63070 SC$=STRING$(8,"0")+MID$(ST
R$(SC),2)
63080 SC$=RIGHT$(SC$,7)
63090 SC$=SC$+"   "
63100 NH$=SC$+N$
63110 HS$=LEFT$(HS$,N)+NH$+MID$(
HS$,N,181-N)
63120 FOR N=1 TO 200 STEP 20:PRI
NT MID$(HS$,N,20):NEXT N
63130 RETURN
63900 PS=PEEK(27)*256+PEEK(28)-2
51
63910 FOR N=1 TO 201:HS$=HS$+CHR
$(PEEK(PS+N)):NEXT N:RETURN
```

```
63920 FOR N=1 TO LEN(HS$):POKE P
S+N,ASC(MID$(HS$,N,1)):NEXT N:CS
AVE "NOMBRE"
63999 REM90000000    PACO    800
00000  SEBASTIAN 70000000  NICOL
AS   60000000   ISABEL   500000
00  CATALINA  40000000   MIGUEL
  30000000  ROBERTO   20000000
JOAQUIN   10000000   MARIA   00
000001   ALFREDO.................
.................................
```

Listado completo del «Parabrisas»

```
850 PCLEAR 8:PMODE 3,1:PCLS:GOTO
 890
855 PMODE 3,5:FORN=2TO21 STEP 2:
LINE(0,0)-(255,191),PSET,B
860 DRAW "S"+STR$(N)+"BM128,96;B
M-14,-20;R28D40L28U40D16R28D2L28
D10R28BM-4,+4;L4D4R4U4BM-16,+0;L
4D4R4U4BM+2,+0;R8BM+0,+2;L8BM+0,
+2;R8BM+8,+4;D4L4U4L16D4L4U4BM+0
,-12;U4R6D4U4L2U2L2D2L2D4R1U2R4D
2L3U2"
865 PCOPY 6 TO 2:PCOPY 7 TO 3
870 PLAY"V"+STR$(N+10)+"O1T255CD
CD"
875 PCLS:NEXT N
880 PMODE 3,1:FOR N=1TO200 STEP
10:PLAY "T"+STR$(255-N)+"O1C05B0
3C":CIRCLE(100,145),N,2,0.5,0.5,
0:NEXT N
885 FOR N=60 TO 1 STEP-1:PLAY"V"
+STR$(INT(N/2))+"O1T255CDCDCDCD"
:CIRCLE(128,96),(N*3),4:NEXT N:R
UN
890 PMODE 1,1:PCLS 2:LINE (80,40
)-(140,80),PRESET,BF:LINE (110,4
0)-(110,0),PRESET:PCOPY 3 TO 2
```

```
895 PMODE 1,4:PCLS3:COLOR 2,3:CI
RCLE(160,105),90,2:CIRCLE(160,10
5),90,2:PAINT(75,105),2,2:LINE(1
57,105)-(163,25),PSET,BF
896 CIRCLE (100,60),20,2,1.5:LIN
E(100,60)-(100,80),PSET:CIRCLE (
160,60),20,2,1.5:LINE(160,60)-(1
70,70),PSET:CIRCLE(220,60),20,2,
1.5:LINE(220,60)-(225,55),PSET:C
IRCLE(20,40),10,2,1.5:LINE(20,40
)-(25,35),PSET:CIRCLE (60,40),10
,2,1.5
897 LINE(60,40)-(70,40),PSET:PMO
DE 3,1:SCREEN 1,0:GOTO 855
```

Listado completo de «Artext»

```
10 GOTO800
15 A$=INKEY$:IF A$="" THEN DRAW"
XCC$;R1;XCA$;L1":GOTO 15:ELSE 20
0
25 DRAW"C"+CB$+"A"+D$+"S"+STR$(S
):ON (ASC(C$)-32) GOSUB33,34,35,
36,37,38,39,40,41,42,43,44,45,46
,47,48,49,50,51,52,53,54,55,56,5
7,58,59,60,61,62,63,64,65,66,67,
68,69,70,71,72,73,74,75,76,77,78
,79,80,81,82,83,84,25,86,87,88,8
9,90:RETURN
33 RETURN
34 DRAW"BM+0,-6DBM+2,+0UBM+4,+6"
:RETURN
35 RETURN
36 RETURN
37 RETURN
38 RETURN
39 DRAW"BM+0,-6DBM+4,+5":RETURN
40 DRAW"BM+2,+0HU4EBM+4,+6":RETU
RN
```

```
41 DRAW"BM+1,+0EU4HBM+5,+6":RETU
RN
42 DRAW"BM+0,-1E4BM+0,+4H4BM+8,+
5":RETURN
43 DRAW"BM+0,-3R4L2U2D4BM+5,+1":
RETURN
44 DRAW"BM-1,+0DGBM+4,-2":RETURN
45 DRAW"BM+0,-3R4BM+4,+3":RETURN
46 DRAW"BM-1,+0UBM+4,+1":RETURN
47 DRAW"BM+0,-1E4BM+4,+5":RETURN
48 DRAW"BM+0,-1FR2EU4HL2GD4BM+8,
+1":RETURN
49 DRAW"BM+1,+0U6GBM+6,+5":RETUR
N
50 DRAW"BM+4,+0L4UER2EU2HL2GBM+8
,+5":RETURN
51 DRAW"BM+0,-1FR2EUHL2R2EUHL2GB
M+8,+5":RETURN
52 DRAW"BM+3,+0U6G3R4BM+4,+3":RE
TURN
53 DRAW"BM+0,-1FR2EU2HL3U2R4BM+4
,+6":RETURN
54 DRAW"BM+0,-2ER2FDGL2HU4ER2FBM
+4,+5":RETURN
55 DRAW"BM+2,+0U2E2U2L4BM+8,+6":
RETURN


56 DRAW"BM+1,+0R2EUHL2HUER2FDGL2
GDFBM+7,+0":RETURN
57 DRAW"BM+0,-1FR2EU4HL2GDFR3BM+
4,+3":RETURN
58 DRAW"BM+0,-5DBM+0,+2DBM+4,+1"
:RETURN
59 DRAW"BM+0,-5DBM+0,+2DGBM+5,+0
":RETURN
60 RETURN
61 DRAW"BM+0,-2R4BM+0,-2L4BM+8,+
4":RETURN
62 RETURN
```

```
63 DRAW"BM+2,+0UBM+0,-1UREUHLGBM
+7,+5":RETURN
64 REM
65 DRAW"U5ER2FD5U3L4BM+8,+3":RET
URN
66 DRAW"U6R3FDGFDGL3U3R3BM+5,+3"
:RETURN
67 DRAW"BM+1,+0HU4ER2FHL2GD4FR2E
BM+4,+1":RETURN
68 DRAW"U6R3FD4GL3BM+8,+0":RETUR
N
69 DRAW"R4L4U3R4L4U3R4BM+4,+6":R
ETURN
70 DRAW"U3R4L4U3R4BM+4,+6":RETUR
N
71 DRAW"BM+1,+0R2EULRDGL2HU4ER2F
BM+4,+5":RETURN
72 DRAW"U6D3R4U3D6BM+4,+0":RETUR
N
73 DRAW"BM+1,+0R2LU6LR2BM+4,+6":
RETURN
74 DRAW"BM+0,-1FR2EU5BM+4,+6":RE
TURN
75 DRAW"U6BM+0,+3RE3G3F3BM+4,+0"
:RETURN
76 DRAW"R4L4U6BM+8,+6":RETURN
77 DRAW"U6F2E2D6BM+4,+0":RETURN
78 DRAW"U6DF4DU6BM+4,+6":RETURN
79 DRAW"BM+1,+0R2EU4HL2GD4FBM+7,
+0":RETURN
80 DRAW"U6R3FDGL3BM+8,+3":RETURN
81 DRAW"BM+1,+0R2EU4HL2GD4FBM+1,
-2F2BM+4,+0":RETURN
82 DRAW"U6R3FDGL3RF3BM+4,+0":RET
URN
83 DRAW"BM+0,-1FR2EH4ER2FBM+4,+5
":RETURN
84 DRAW"BM+2,+0U6L2R4BM+4,+6":RE
TURN
85 DRAW"BM+0,-6D5FR2EU5BM+4,+6":
RETURN
```

```
86 DRAW"BM+0,-6D4F2E2U4BM+4,+6":
RETURN
87 DRAW"BM+0,-6D6E2F2U6BM+4,+6":
RETURN
88 DRAW"UE4UBM+0,+6UH4UBM+8,+6":
RETURN
89 DRAW"BM+2,+0U4H2F2E2BM+4,+6":
RETURN
90 DRAW"R4L4UE4UL4BM+8,+6":RETUR
N
200 IF INSTR(1,B$,A$)=0 THEN SOU
ND2,5:GOTO 15
210 A=ASC(A$):IF A=64 THEN 300
220 IF A=83 AND S<45 THEN S=S+4:
GOTO 290
230 IF A=88 THEN S=4:GOTO 290
240 IF A=67 THEN 500
250 IF A=80 THEN 600


270 IF A=94 THEN 700
280 IF A>47 AND A<57 THEN CB$="C
"+A$ ELSE DRAW"C"+CB$+"A"+D$+"S"
+STR$(S)+A$
290 SOUND (89+ASC(A$)),2:GOTO 15
300 C$=INKEY$:IF C$="" THEN DRAW
"BM+2,+2;XCC$;R2L4R2;XCA$;R2L4R2
BM-2,-2":GOTO 300
310 IF C$=" "THEN DRAW"XCA$;U6RD
6RU6RD6RU6RD6BM+3,+0":GOTO 300
320 C=ASC(C$):IF C=64 THEN 15
330 IF C=13 THEN 420
340 IF C=12 THEN 500
350 IF C>32 AND C<91 THEN GOSUB
25:GOTO 300
370 IF C=10 THEN M$="+0,+10"
380 IF C=8 THEN M$="-8,+0"
390 IF C=9 THEN M$="+8,+0"
400 IF C=94 THEN M$="+0,-10"
410 DRAW"S"+STR$(S)+"A"+D$+"BM"+
M$:GOTO 300
```

```
420 D$=INKEY$:IF D$="" THEN SOUN
D 220,2:GOTO 420
430 D=ASC(D$):IF D<48 OR D>51 TH
EN GOTO 420 ELSE 300
500 CLS4:PRINT"BORRAR PANTALLA (
S/N)";:INPUT D$:IF D$<>"S" THEN
1000 ELSE PCLS:GOTO 1000
600 CLS0:PRINT" COORDENADAS PARA
 PAINT";:INPUT P1,P2:PRINT"COLOR
";:INPUT PC:PRINT"COLOR DEL LIMI
TE";:INPUT BC
610 PMODE Z,1:SCREEN 1,Y:PAINT(P
1,P2),PC,BC:GOTO 15
700 CLS4:PRINT @ 5,"RUTINA CASSE
TTE":PRINT @ 69,"S = SAVE":PRINT
 @ 133,"L = LOAD"
710 PRINT @ 197," ";:INPUT T$:IF
 T$<>"S" AND T$<>"L" THEN 710
720 PRINT @ 261,"NOMBRE FICHERO"
;:FOR N=1 TO 7:PRINT CHR$(128);:
NEXT:PRINT @ 298,"  ";:INPUT F$:
IF LEN(F$)>7 THEN PRINT @ 298,,"
DEMASIADO LARGO":GOTO 720:ELSE F
$="M"+F$:PRINT @ 384,"CUANDO CIN
TA LISTA PULSAR BARRA ESPACIADO"
730 IF INKEY$=""THEN 730
740 IF T$="S" THEN PRINT @ 384,"
GUARDAMOS LA PANTALLA":CSAVEMF$,
1535,(1535+(1535*PG)),(1535*PG):
FOR N=1 TO 4:PRINT @ 384,"PANTAL
LA GUARDADA":SOUND 1,5:PRINT @ 3
84," ":SOUND 50,5:NEXT:GOTO 1000
750 IF T$="L" THEN PMODE Z,1:SCR
EEN 1,Y:COLOR CB,CA:CLOADMF$:GOT
O 15
800 CLS:ST=1:PRINT,,"MODOS ALTA
RESOLUCION":PRINT @ 97,"0=128 X
96(";CHR$(128);") DOS COLORES":P
RINT @ 161,"1=128 X 96 (";CHR$(1
28);")CUATRO COLORES":PRINT @ 22
```

```
5,"2=192 X 128(";CHR$(140);") DO
S COLORES":PRINT @ 289,"3=192 X
128(";CHR$(140);")CUATRO COLORES
"
810 PRINT @ 321,,," 4=256 X 192(
";CHR$(142);") DOS COLORES":PRIN
T @ 455,"RESOLUCION REQUERIDA";
820 INPUT Z:IF Z>4 THEN PRINT @
450,"INCORRECTA";:GOTO 830
830 IF Z=0 OR Z=2 OR Z=4 THEN 1
020
900 CLS:PRINT "MODO CUATRO COLOR
ES",,,"CONJUNTO COLORES 1",,,,"1
=";CHR$(143);"VERDE","2=";CHR$(1
59);"AMARILLO","3=";CHR$(175);"A
ZUL","4=";CHR$(191);"ROJO"
910 PRINT,,,"CONJUNTO COLORES 2
",,,,"5=";CHR$(207);"CANELA","6=
";CHR$(223);"CIAN","7=";CHR$(239
);"MAGENTA","8=";CHR$(255);"NARA
NJA"
920 PRINT @ 416,"COLOR DEL FONDO
";:INPUT CA$:PRINT "COLOR PRIMER
 PLANO";:INPUT CB$
930 CA=VAL(CA$):CB=VAL(CB$):IF A
BS(CA-CB)>3 OR CA=0 OR CB=0 THEN
 PRINT @ 480,"COMBINACION INCORR
ECTA";:GOTO 920
940 IF CA<5 THEN Y=0 ELSE Y=1
950 CC$="C"+STR$(Y)
1000 PMODE Z,1:SCREEN 1,Y:COLOR
 CB,CA::IF ST=1 THEN PCLS:ST=0
1009 REM INICIALIZAR VARIABLES
1010 CA$="C"+CA$:CB$="C"+CB$:PG
$="12244":PG=VAL(MID$(PG$,Z+1,1
)):B$="ICUDLREFGHSXP012345678@^
":S=4:D$="0":DRAW"S4BM10,10":GO
TO 15
1020 CLS:PRINT"MODO DOS COLORES"
,,,,,,"CONJUNTO COLORES 1",,,,"0
```

```
=";CHR$(128);"NEGRO","1=";CHR$(1
43);"VERDE"
1030 PRINT,,,,"CONJUNTO COLORES
2",,,,"0=";CHR$(128);"NEGRO","5=
";CHR$(207);"CANELA"
1040 PRINT @ 416,"COLOR DEL FOND
O";:INPUT CA$:PRINT "COLOR PRIME
R PLANO";:INPUT CB$
1050 CA=VAL(CA$):CB=VAL(CB$):IF
 CA=1 OR CA=5 AND CB=0 THEN 108
0
1060 IF CB=1 OR CB=5 AND CA=0 T
HEN 1080
1070 PRINT @ 480,"COMBINACION IN
CORRECTA";:GOTO 1040
1080 IF CA=1 OR CB=1 THEN Y=0:C
C$="C"+CB$:GOTO 1000
1090 Y=1:CC$="C"+CB$:GOTO 1000
```

Listado completo de «Superaterrizaje»

```
10 GOTO 100
20 FOR N=1 TO LEN(WD$):C$=MID$(W
D$,N,1):GOSUB 25:NEXT N:RETURN
25 ON(ASC(C$)-31)GOSUB32,33,34,3
5,36,37,38,39,40,41,42,43,44,45,
46,47,48,49,50,51,52,53,54,55,56
,57,58,59,60,61,62,63,64,65,66,6
7,68,69,70,71,72,73,74,75,76,77,
78,79,80,81,82,83,84,85,86,87,88
,89,90:RETURN
32 DRAW"BM+8,+0":RETURN
33 RETURN
34 DRAW"BM+0,-6DBM+2,+0UBM+4,+6"
:RETURN
35 RETURN
36 RETURN
37 RETURN
38 RETURN
```

```
39 DRAW"BM+0,-6DBM+4,+5":RETURN
40 DRAW"BM+2,+0HU4EBM+4,+6":RETU
RN
41 DRAW"BM+1,+0EU4HBM+5,+6":RETU
RN
42 DRAW"BM+0,-1E4BM+0,+4H4BM+8,+
5":RETURN
43 DRAW"BM+0,-3R4L2U2D4BM+5,+1":
RETURN
44 DRAW"BM-1,+0DGBM+4,-2":RETURN
45 DRAW"BM+0,-3R4BM+4,+3":RETURN
46 DRAW"BM-1,+0UBM+4,+1":RETURN
47 DRAW"BM+0,-1E4BM+4,+5":RETURN
48 DRAW"BM+0,-1FR2EU4HL2GD4BM+8,
+1":RETURN
49 DRAW"BM+1,+0U6GBM+6,+5":RETUR
N
50 DRAW"BM+4,+0L4UER2EU2HL2GBM+8
,+5":RETURN
51 DRAW"BM+0,-1FR2EUHL2R2EUHL2GB
M+8,+5":RETURN
52 DRAW"BM+3,+0U6G3R4BM+4,+3":RE
TURN
53 DRAW"BM+0,-1FR2EU2HL3U2R4BM+4
,+6":RETURN
54 DRAW"BM+0,-2ER2FDGL2HU4ER2FBM
+4,+5":RETURN
55 DRAW"BM+2,+0U2E2U2L4BM+8,+6":
RETURN
56 DRAW"BM+1,+0R2EUHL2HUER2FDGL2
GDFBM+7,+0":RETURN
57 DRAW"BM+0,-1FR2EU4HL2GDFR3BM+
4,+3":RETURN
58 DRAW"BM+0,-5DBM+0,+2DBM+4,+1"
:RETURN
59 DRAW"BM+0,-5DBM+0,+2DGBM+5,+0
":RETURN
60 RETURN
61 DRAW"BM+0,-2R4BM+0,-2L4BM+8,+
4":RETURN
```

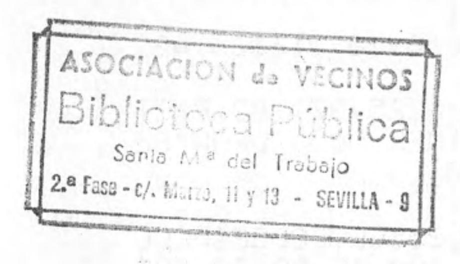
```
62 RETURN
63 DRAW"BM+2,+0UBM+0,-1UREUHLGBM
+7,+5":RETURN
64 REM
65 DRAW"U5ER2FD5U3L4BM+8,+3":RET
URN
66 DRAW"U6R3FDGFDGL3U3R3BM+5,+3"
:RETURN
67 DRAW"BM+1,+0HU4ER2FHL2GD4FR2E
BM+4,+1":RETURN
68 DRAW"U6R3FD4GL3BM+8,+0":RETUR
N
69 DRAW"R4L4U3R4L4U3R4BM+4,+6":R
ETURN
70 DRAW"U3R4L4U3R4BM+4,+6":RETUR
N
71 DRAW"BM+1,+0R2EULRDGL2HU4ER2F
BM+4,+5":RETURN
72 DRAW"U6D3R4U3D6BM+4,+0":RETUR
N
73 DRAW"BM+1,+0R2LU6LR2BM+4,+6":
RETURN
74 DRAW"BM+0,-1FR2EU5BM+4,+6":RE
TURN
75 DRAW"U6BM+0,+3RE3G3F3BM+4,+0"
:RETURN
76 DRAW"R4L4U6BM+8,+6":RETURN
77 DRAW"U6F2E2D6BM+4,+0":RETURN
78 DRAW"U6DF4DU6BM+4,+6":RETURN
79 DRAW"BM+1,+0R2EU4HL2GD4FBM+7,
+0":RETURN
80 DRAW"U6R3FDGL3BM+9,+3":RETURN
81 DRAW"BM+1,+0R2EU4HL2GD4FBM+1,
-2F2BM+4,+0":RETURN
82 DRAW"U6R3FDGL3RF3BM+4,+0":RET
URN
83 DRAW"BM+0,-1FR2EH4ER2FBM+4,+5
":RETURN
84 DRAW"BM+2,+0U6L2R4BM+4,+6":RE
TURN
```

```
85 DRAW"BM+0,-6D5FR2EU5BM+4,+6":
RETURN
86 DRAW"BM+0,-6D4F2E2U4BM+4,+6":
RETURN
87 DRAW"BM+0,-6D6E2F2U6BM+4,+6":
RETURN
88 DRAW"UE4UBM+0,+6UH4UBM+8,+6":
RETURN
89 DRAW"BM+2,+0U4H2F2E2BM+4,+6":
RETURN
90 DRAW"R4L4UE4UL4BM+8,+6":RETUR
N
100 PMODE 3,1:SCREEN 1,1:PCLS:J0
=10:J1=70:T0=J0:T1=J1:POKE&HFFD7
,0
110 DIM A1(0,9):DIM A2(0,3):DIM
A3(0,20)
120 DRAW"C8BM8,8D5R5U5L2U4D4L2D5
L2D3R8U2L2"
130 GET(4,4)-(14,16),A2,G
140 PCLS
150 LINE(0,20)-(50,50),PSET:LINE
(50,50)-(60,40),PSET:LINE(60,40)
-(120,70),PSET:LINE(120,70)-(120
,120),PSET:LINE(120,120)-(180,13
5),PSET:LINE(180,135)-(255,30),P
SET:PAINT(10,10),6,8
160 LINE(0,100)-(50,100),PSET:LI
NE(50,100)-(60,120),PSET:LINE(60
,120)-(120,150),PSET:LINE(120,15
0)-(190,160),PSET:LINE(190,160)-
(230,110),PSET:LINE(230,110)-(23
0,140),PSET:LINE(230,140)-(245,1
40),PSET:LINE(245,140)-(245,100)
,PSET
170 LINE(245,100)-(255,80),PSET:
PAINT(255,100),6,8:LINE(230,140)
-(245,145),PSET,B:PAINT(235,142)
,7,8
175 WD$="ATERRIZAJE":DRAW"BM10,1
85"+"S8":GOSUB 20:WD$="DA/OS":DR
```

```
AW"C1BM90,15":GOSUB 20:WD$="TIEM
PO":DRAW"BM116,35":GOSUB 20:TIME
R=0
180 PMODE4,1:SCREEN1,1
190 LR=LR+(JOYSTK(0)<20)-(JOYSTK
(0)>40):UD=UD+(JOYSTK(1)<20)-(JO
YSTK(1)>40)
200 J0=J0+SGN(LR):J1=J1+(2*(SGN(
UD))):J1=J1+1:IF J0>228 AND J0<2
32 AND J1>128 AND J1<132 THEN 26
0
210 IF PPOINT(J0,J1)>0 OR PPOINT
(J0+11,J1)>0 OR PPOINT(J0,J1+13)
>0 OR PPOINT(J0+11,J1+13)>0 THEN
 GOSUB 250
220 PUT(T0,T1)-(T0+12,T1+12),A1,
PSET:T0=J0:T1=J1
230 PUT(J0,J1)-(J0+10,J1+12),A2,
OR
235 PUT(190,22)-(225,37),A3,PSET
:WD$=STR$(INT(TIMER/50)):DRAW"BM
180,35":GOSUB 20
240 GOTO 190
250 PLAY"O1T255L255CDCDCDGECDCDC
":DA=DA+1:GOSUB 300:IF DA<20 THE
N RETURN ELSE FOR N=1 TO 15 STEP
 2:PLAY"V"+STR$(N*2)+"CCDD":CIRC
LE(J0+5,J1+6),N,1:NEXT N:FOR N=1
5 TO 1 STEP -1:CIRCLE(J0+5,J1+6)
,N,0:NEXT N:RUN
260 PRINT "EXITO":END
300 PUT(190,2)-(225,17),A3,PSET:
WD$=STR$(DA):DRAW"BM190,15":GOSU
B 20:RETURN
```

AHORA ES SU TURNO

Ahora sí que realmente es su turno. No tenemos espacio para recoger más ideas. Depende de usted el demostrar al mundo lo buen programador que es. Hemos intentado cubrir los aspectos que a nuestro parecer son más importantes en la programación de juegos. ¿Qué tal si los reúne todos y prepara algo realmente importante? Buena suerte y que se divierta con sus juegos para el Dragón.



ediciones
técnicas



REDE

BARCELONA
(ESPAÑA)